Quaderni di informatica

2

Rassegna di tecnologia ed applicazioni degli elaboratori elettronici - Anno I - Numero 2 - Agosto 1974

ARCHIVIO STORICO
ASSOCIAZIONE POZZO DI MIELE
FPDM-28
RQDI 100

Honeywell
Honeywell Information Systems Italia

FDD9-25

Quaderni di informatica

Rassegna di tecnologia ed applicazioni degli elaboratori elettronici
Anno I - Numero 2 - Agosto 1974

Sommario

Introduzione al firmware
Oltre che dei tradizionali « hardware » e « software », i moderni elaboratori elettronici sono costituiti anche di « firmware ». Quali concetti cela realmente questo neologismo? E quali significati pratici esso comporta? A queste domande risponde il presente articolo, sulla base anche di una significativa esperienza italiana.
A. BRIOSCHI

La correttezza dei programmi
La ricerca e la correzione degli errori dei programmi costituiscono, notoriamente, un pesante onere nella realizzazione del software. Esiste la possibilità di controllare questo problema, fondamentale nell'impiego degli elaboratori elettronici? Questo articolo espone, in termini piani, le idee attualmente sviluppate per cercare di dare una soluzione a questo difficile problema, che ha profonde implicazioni concettuali.
G. DEGLI ANTONI

Il concetto di database
L'organizzazione dell'archivio dei dati costituisce un problema centrale nella utilizzazione dei sistemi di elaborazione elettronica. In questo campo, il « database » rappresenta la concezione più moderna ed il logico superamento dei metodi tradizionali, derivato dal ripensamento, in termini critici, dell'intero problema. Il concetto di « base dei dati », che è, giustamente, considerato una pietra miliare dell'informatica, viene qui presentato nel quadro generale della evoluzione dei metodi di organizzazione e gestione degli archivi di dati.
E. RAJA

La protezione giuridica del software
È il « software » brevettabile? La materia è controversa e si presta a sottili discussioni. Nel presente articolo si fa il punto su questo delicato problema. In un articolo successivo si esaminerà quali forme alternative di protezione giuridica si possano applicare ai programmi degli elaboratori elettronici.
C. FALCETTI

L'elaborazione elettronica nel settore bancario in Italia
L'articolo riassume i risultati più significativi di una recente indagine sulla utilizzazione degli elaboratori elettronici nel settore indicato. Oltre a fornire una aggiornata panoramica statistica, l'articolo mette in luce interessanti correlazioni. Indagini analoghe, riguardanti altri settori di applicazione, verranno presentate regolarmente sulla rivista.
V. MARANDELLA

Resoconti e recensioni

Direttore Responsabile
F. Filippazzi

Comitato di Redazione
A. Cicu, M. De Marco, S. Fabris,
C. Falcetti, R. Levrero, P. Lupo,
N. Minnaja, G. Occhini, G. Rapelli,
F. Ricci, M. Rossi, F. Senzasono

Redazione
Honeywell Information Systems Italia
Centro di Ricerca e Progettazione
20010 Pregnana Milanese (Milano)

Stampa
grafiche A. Nava, Milano

Autorizzazione del Tribunale di Milano
n. 93 del 20 Marzo 1974

Introduzione al firmware

ANTONIO BRIOSCHI
Honeywell Information Systems Italia
Centro di Ricerca e Progettazione di Pregnana Milanese

1. Introduzione

Il termine *« firmware »* fu coniato nel 1967 da Opler [1], ad indicare qualcosa di intermedio tra l'*« hardware »* (la macchina fisica) e il *« software »* (i programmi).

In effetti, il firmware rientra nel concetto di *« microprogrammazione »*, introdotto fin dal 1951 da Wilkes, in una prolusione presentata all'Università di Manchester [2]. Lo scopo principale che egli si proponeva era di « fornire una soluzione matematica ed ordinata per il progetto della unità di controllo di un qualsiasi sistema di elaborazione ».

Occorre ricordare che l'unità di controllo è, in pratica, il « cervello » dell'elaboratore in quanto comanda e coordina il funzionamento di tutte le parti della macchina allo scopo di ottenere determinati risultati. In sostanza, l'unità di controllo interpreta le istruzioni da eseguire, generando delle sequenze di comandi e inviandoli ai vari organi interessati. Nelle soluzioni tradizionali, l'unità di controllo era realizzata mediante un insieme « ad hoc » di reti logiche, sostanzialmente privo di regolarità, e pertanto complicato, rigido e costoso. (Unità di controllo a *« logica cablata »*).

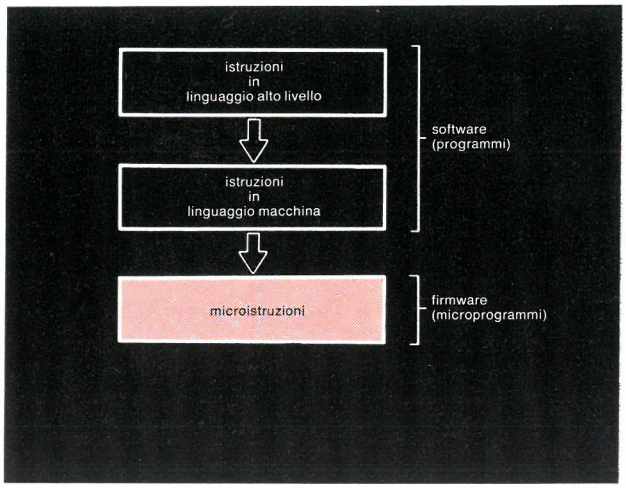
Nella soluzione proposta da Wilkes le *sequenze elementari di comandi* che l'Unità di controllo può emettere vengono registrate in una memoria fissa, da cui vengono estratte quando occorre (Unità di controllo a *« logica memorizzata »*). Queste sequenze elementari di comandi (a ognuna delle quali corrisponde un passo elementare di elaborazione) costituiscono delle *primitive* del calcolatore che Wilkes chiamò *microistruzioni* ed il loro insieme costituisce ciò che oggi viene designato come *firmware.*

La relazione che esiste tra il concetto di microistruzione e quello usuale di istruzione è illustrata schematicamente in fig. 1. Ogni istruzione (in linguaggio macchina) è realizzata da uno o più microprogrammi, ciascuno dei quali è costituito da un certo insieme di microistruzioni. In altri termini, il firmware risulta essere un intermediario tra il software e l'hardware, che nelle concezioni precedenti comunicavano direttamente, nel senso che ogni istruzione veniva direttamente interpretata dall'hardware mediante una specifica rete logica (fig. 2).

Le idee di Wilkes rimasero per parecchi anni sostanzialmente ad un livello accademico, in quanto non esisteva una soluzione fisica adeguata, in termini di costo e di prestazioni, per la memoria fissa su cui la soluzione era basata. Il progresso della tecnologia elettronica e, in particolare, il recente avvento dei circuiti integrati su larga scala (LSI), hanno portato alla ribalta questi concetti in termini pratici e applicativi.

In effetti, l'introduzione della microprogrammazione ha una portata che va ben oltre la semplice razionalizzazione concettuale del progetto dei sistemi di elaborazione. Essa costituisce, infatti, lo strumento per un progresso effettivo, sul piano pratico, di aspetti fondamentali quali la possibilità di *emulazione* (su cui è basato il concetto di famiglie di

Fig. 1 - Il concetto di microistruzione come sottomodulo dell'istruzione.



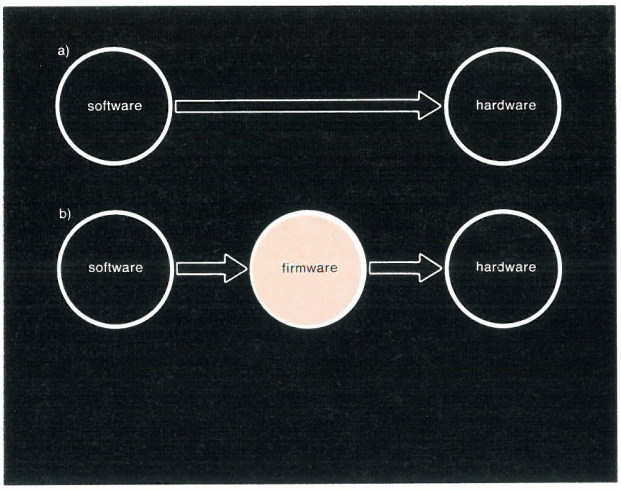


Fig. 2 - Componenti fondamentali dei sistemi convenzionali (a) e di quelli microprogrammati (b).
Nel primo caso, le istruzioni che costituiscono i programmi (contenuti nella memoria principale) sono tradotte dall'hardware direttamente in segnali operativi per i vari organi dell'elaboratore.
Nel secondo caso, le istruzioni sono interpretate in termini di microistruzioni (residenti nella memoria di controllo), e queste tradotte dall'hardware in segnali operativi.

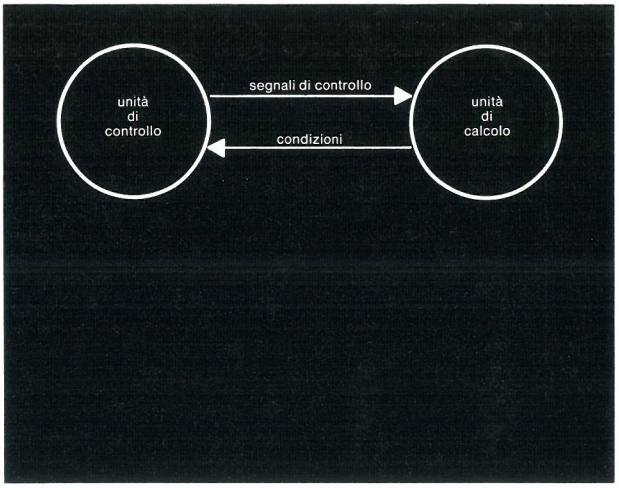


Fig. 3 - Interazioni tra unità di controllo e unità di calcolo.

elaboratori compatibili), il miglioramento della *affidabilità*, la capacità di *autodiagnosi*.
La definizione del firmware, ossia — in sostanza — la scelta delle sequenze elementari di comando dell'elaboratore, risulta il punto chiave sia per lo sfruttamento ottimale di quanto offre man mano la tecnologia sia per la migliore strutturazione dei programmi da dare in corredo al sistema.
Il firmware si affianca, pertanto, come nuova entità fondamentale, all'hardware e al software, aprendo prospettive nuove alla ottimizzazione del disegno architettonico dei sistemi di elaborazione e, quindi, al loro impiego pratico.

2. Unità di controllo e unità di calcolo

Consideriamo un sistema di elaborazione dati qualsiasi e chiamiamolo sistema A. La sua funzionalità è definita dall'insieme delle istruzioni, dai tipi di memoria (memoria operativa, dischi, ecc.), dai canali e così via. I programmi scritti in linguaggio macchina di questo sistema possono essere eseguiti su di esso direttamente.
Supponiamo ora di avere dei programmi, in linguaggio macchina, scritti per un altro sistema, che chiamiamo B, e di volerli fare eseguire dal sistema A. Per far ciò è necessario scrivere un certo numero di programmi che ricreino sul sistema A il modo di funzionamento del sistema B così da avere gli stessi risultati quando i programmi di quest'ultimo vengono eseguiti. La combinazione del sistema A e dei suoi programmi si dice definire un *sistema virtuale* che equivale funzionalmente al sistema B. Il sistema A in tal caso è detto *sistema ospite* e il sistema B *sistema di targa*.
Il processo fin qui descritto è detto *simulazione* del sistema di targa ed è realizzato essenzialmente mediante aggiunte e modifiche alla funzionalità del sistema ospite. Questo processo però non è sempre possibile perché la possibilità di costruzione di un qualsiasi sistema virtuale è funzione del tipo e della quantità di risorse disponibili sul sistema ospite.
Il concetto di sistema ospite, così come è stato introdotto, può essere utilizzato per analizzare anche il comportamento interno di una macchina. Per fare questa analisi cominciamo col dividere il sistema in due parti: le Unità Periferiche e l'Unità Centrale. Consideriamo solo quest'ultima, in quanto si può ritenere che in essa sia concentrata tutta la capacità di elaborazione del sistema. Idealmente l'Unità Centrale è composta di due parti: una *unità di controllo* ed una *unità di calcolo*.
L'unità di calcolo è quella che detiene tutte le risorse di memorizzazione e trasformazione delle informazioni; così definita, l'unità di calcolo è costituita, in pratica, dalla memoria operativa, da tutte le reti (aritmetiche, logiche, di spostamento, ecc.) e da tutti i registri. L'unità di controllo è quella che dirige le attività della unità di calcolo generando una sequenza ordinata di processi le cui computazioni sono visibili globalmente all'esterno come *istruzioni di macchina*.
Sostanzialmente l'esecuzione di una istruzione avviene in tre passi fondamentali. 1) L'unità di controllo comanda l'estrazione dalla memoria operativa, e la memorizzazione nei vari registri, della istruzione. 2) L'unità di controllo (cui è inviato il codice di funzione) riconosce il tipo di istruzione. 3) In base al riconoscimento fatto, essa attiva il processo che esegue tutte le operazioni necessarie per la esecuzione della istruzione.
Da quanto detto risulta chiaro che, fondamentalmente le due unità interagiscono fra di loro scambiandosi messaggi il cui contenuto è, alternativamente, segnali di controllo ed informazioni riguar-

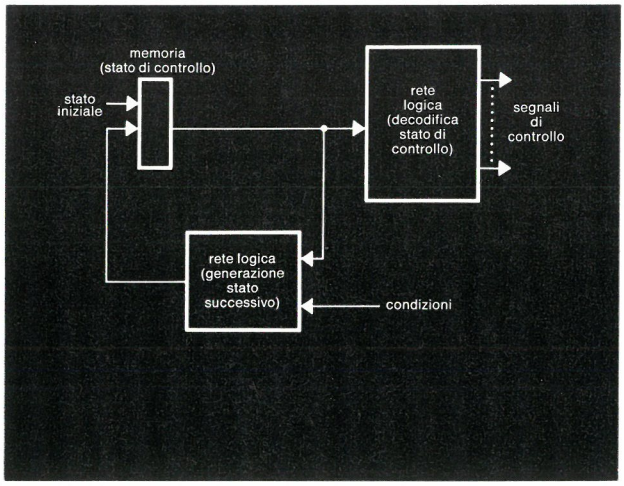


Fig. 4 - Unità di controllo.

danti lo stato della unità di calcolo, come mostrato in fig. 3.

L'unità di calcolo così enucleata può essere considerata il sistema ospite a livello hardware. La funzionalità di questo sistema può essere descritta associando ad ogni segnale di controllo la computazione eseguita dalla unità di calcolo. L'insieme di queste computazioni definisce il linguaggio macchina del sistema ospite a livello harwdare. I segnali di controllo sono forniti dalle reti della unità di controllo in funzione della interpretazione che esse fanno delle informazioni che fornisce il mondo esterno. E' intuitivo che l'insieme delle due unità costituisce un sistema virtuale a livello hardware.

E' essenziale rilevare che cambiando solo l'unità di controllo si può realizzare sul sistema ospite un qualsiasi sistema virtuale che simuli un certo sistema di targa.

3. Unità di controllo microprogrammata

Da quanto ora detto deriva l'importanza di realizzare l'unità di controllo in modo che sia facilmente modificabile e adattabile a nuove esigenze.

Per renderci conto del funzionamento di un sistema microprogrammato, esaminiamo più da vicino il funzionamento elementare dell'unità di controllo.

Ricollegandoci a quanto detto nel paragrafo precedente, risulta che il compito dell'unità di controllo è quello di generare per ogni istruzione di macchina una computazione finita il cui risultato è la elaborazione definita della istruzione stessa.

La computazione in oggetto non è, come sembrerebbe all'esterno del sistema, un fenomeno continuo ma un fenomeno discreto composto di una serie di passi. (Ciò si collega, in sostanza, alla natura discreta (binaria) dei circuiti da cui le macchine « digitali » sono costituite). Ogni passo è eseguito in un tempo ben preciso detto *ciclo di macchina;* questo tempo è scandito da un orologio interno alla macchina stessa. E' fondamentale notare che ad ogni passo il tempo riparte sempre dall'origine, ciò fa sì che l'elaborazione iniziale di ogni passo sia sempre uguale e si differenzia solo dopo l'accesso alle varie memorie del sistema. Queste memorie sono il solo mezzo che permette un passaggio di informazione tra un passo e il successivo.

In ogni passo della computazione o *« stato di controllo »* vengono generati tutti i segnali di controllo che servono a portare a termine una certa operazione che è richiesta dalla computazione in corso e un'altra serie di segnali per stabilire quale sarà il prossimo stato di controllo da eseguire.

Da tutto ciò risulta chiaro che le funzioni che l'unità di controllo deve eseguire sono due. La prima è la generazione dei segnali di controllo che dirigono l'attività della unità di calcolo e la seconda è la determinazione del successivo stato di controllo da eseguire.

Perché l'unità di controllo possa espletare queste funzioni deve avere almeno queste reti: una memoria per contenere l'informazione che caratterizza lo stato di controllo attuale (codice di stato); una rete di generazione delle informazioni caratterizzanti il successivo stato di controllo, e una rete di decodifica di queste informazioni per ricavare da esse i segnali di controllo (*). Lo schema con le interconnessioni è rappresentato in fig. 4.

Dal modo di funzionamento di questa unità di controllo si deduce che il simulatore hardware della macchina di targa da essa realizzato è un « interpretatore ». Questo è il nome che si dà quando il processo di simulazione viene fatto avanzare istruzione per istruzione, cioè quando una nuova istruzione non viene estratta dalla memoria centrale ed eseguita prima che gli effetti della istruzione in corso non siano stati completamente valutati.

Nelle soluzioni tradizionali l'unità di controllo schematizzata nella fig. 4 era realizzata, a parte la piccola memoria dello stato di controllo, con un insieme di reti logiche interconnesse in modo specifico per ottenere tutte le informazioni di controllo richieste *(logica cablata)*. Tale insieme era pertanto complicato, irregolare, privo di modularità, e, quindi, rigido rispetto alle possibilità di cambiamenti. Nella soluzione microprogrammata, questo insieme di reti logiche viene sostituito sostanzialmente da una memoria *(logica memorizzata)*. In questa memoria *(memoria di controllo)* vengono immagazzinate sia le informazioni necessarie per definire, ad ogni passo, la sequenza dei segnali di controllo da

(*) La presenza della rete di decodifica per la generazione dei segnali di controllo non è concettualmente necessaria perché si potrebbe pensare di connettere direttamente ogni elemento della memoria alla corrispondente linea che porta il segnale alla unità di calcolo. Ciò non viene fatto solo per ragioni di costo. In un calcolatore il numero dei segnali di controllo è dell'ordine delle centinaia, e quindi occorrerebbe un numero pari di elementi di memoria per contenerli, ma considerando che le combinazioni di questi, utilizzate in ogni passo è dell'ordine della decina, risulta chiaro che conviene codificare queste informazioni (in modo da avere una memoria con solo poche decine di elementi) e inserire poi una rete di decodifica che, ad ogni passo, ricava dallo stato di controllo l'insieme dei segnali di controllo.

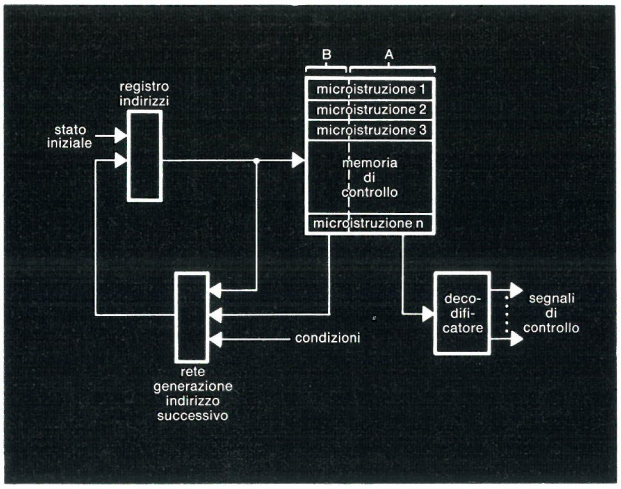


Fig. 5 - Unità di controllo microprogrammata.

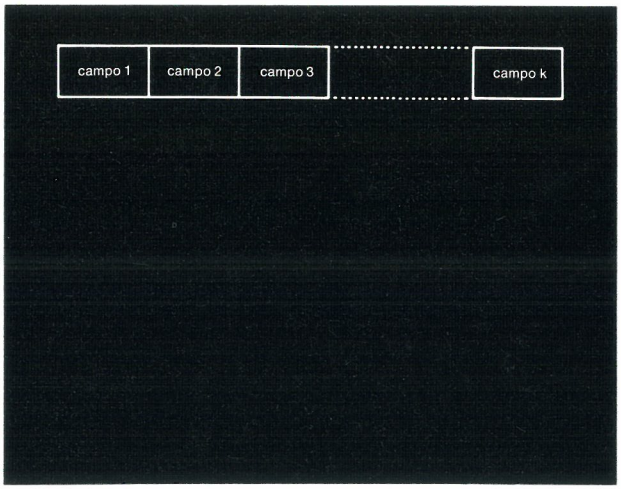


Fig. 6 - Struttura della microistruzione.

emettere, sia le informazioni che servono per determinare il successivo stato di controllo.
In connessione con l'impiego di un dispositivo tipicamente regolare e dotato di modularità quale una matrice di memoria, si ottiene con questa soluzione una razionalizzazione concettuale dell'intera funzione di controllo.
Infatti, mentre nelle soluzioni a logica cablata esiste una relazione rigida tra l'istruzione e la sua realizzazione harware, nella soluzione a logica memorizzata viene introdotto, in corrispondenza con la modularità fisica della memoria di controllo, *un ulteriore livello di modularità: la microistruzione.*
Ciascuna linea della memoria di controllo contiene infatti una sequenza elementare di comandi (corrispondenti cioè ad un ciclo di macchina); queste sequenze elementari, definite in modo da realizzare una elevata ripetitività di impiego, costituiscono dei sottomoduli delle istruzioni e vengono perciò chiamate *microistruzioni.* L'insieme dei microprogrammi costituisce il *firmware* del sistema.
Lo schema di una unità di controllo microprogrammata è illustrato nella fig 5.
Il contenuto dell'indirizzatore della memoria di controllo identifica lo stato di controllo. Ad esso corrisponde una linea della memoria di controllo, il cui contenuto costituisce una microistruzione. Più precisamente, la memoria si può considerare suddivisa in due parti, una (A) che fornisce i segnali di controllo del ciclo, ed una (B) che fornisce le informazioni per il proseguimento della computazione (ossia per la determinazione dell'indirizzo successivo a cui la memoria di controllo deve essere letta).
In questa struttura, dunque, ogni stato di controllo è identificato da un indirizzo appartenente all'insieme di indirizzi validi della memoria di controllo. La dimensione di questo insieme è quindi il numero di stati di controllo che può assumere il sistema.

4. Struttura della microistruzione

Per ragioni analoghe a quelle esposte in nota nel paragrafo precedente, la microistruzione viene codificata. Il processo di codifica si svolge come segue. Per prima cosa si analizzano tutti i segnali di controllo e si raggruppano in insiemi costituiti da segnali mutuamente esclusivi, cioè in insiemi dove un solo segnale alla volta può essere attivato. Ad ogni insieme così individuato si associa un campo della microistruzione. Poiché il numero di campi così ottenuto risulta ancora abbastanza elevato da non poter essere contenuto totalmente in una microistruzione (la cui lunghezza è prefissata con criteri ottimali) questi campi vengono raggruppati in funzioni omogenee ed associati ad un particolare tipo di microistruzione. La generica microistruzione così ottenuta è rappresentata in fig. 6.
Il campo 1 è detto codice di funzione ed è il più importante perché oltre a stabilire la funzione principale della microistruzione ne stabilisce anche il formato cioè numero e dimensione dei campi che seguono. Il numero di campi da cui è composta la microistruzione dà una misura approssimativa del parallelismo dell'unità di calcolo, cioè del numero di operazioni che essa può condurre in parallelo.
Fino a qui si è visto come la microistruzione definisce, in ogni stato di controllo, la attivazione o meno di ogni linea di controllo. Questa non è la sola funzione che deve essere eseguita ad ogni passo. Come si è già detto, esiste un'altra funzione fondamentale che è il calcolo dell'indirizzo (che equivale allo stato di controllo) a cui estrarre la prossima microistruzione. La cosa più semplice per svolgere questa funzione è predisporre una rete che incrementi l'indirizzo di una unità in modo che alla fine della esecuzione della microistruizone in corso l'indirizzo della prossima sia già pronto nell'indirizzatore. Questa struttura però non è sufficiente a soddisfare tutte le esigenze di microprogrammazione perché essa definisce sempre in modo univoco come prossimo stato di controllo quello immediatamente successivo, senza tener conto di nessuna condizione che il mondo esterno fornisce durante l'esecuzione della microistruzione. Queste condizioni inve-

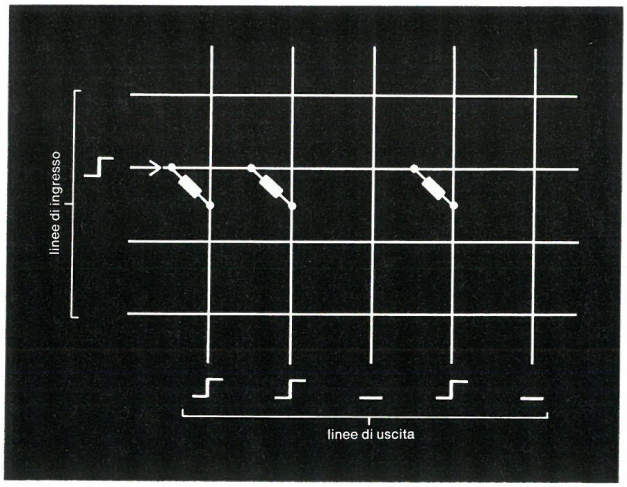


Fig. 7 - Schema di principio di una memoria di sola lettura (ROM).

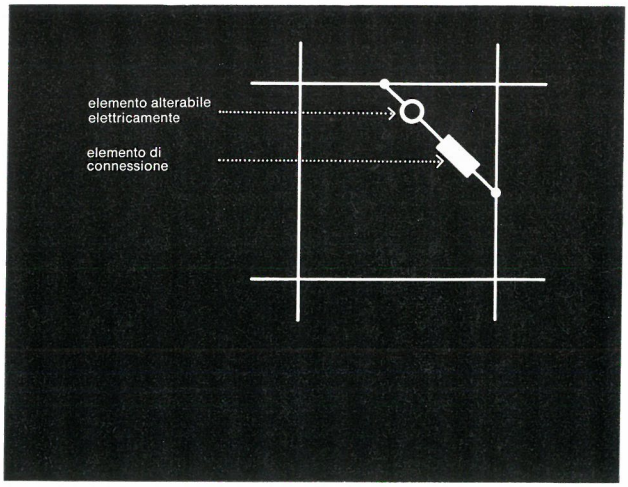


Fig. 8 - Principio delle memorie di sola lettura programmabili (PROM).

ce sono fondamentali perché solo tenendo conto di esse si riesce a controllare l'unità di calcolo. Quindi è fondamentale dotare le microistruzioni di una *funzione di salto,* cioè della capacità di variare la sequenza di esecuzione in funzione di stimoli esterni. La tecnica più comunemente adottata è quella di riservare un campo della microistruzione per questa funzione. In questo campo viene messa una quantità da sommare o meno all'indirizzo attuale, in funzione del valore della condizione esterna.

La rete di generazione dell'indirizzo successivo, indicata nello schema di fig. 5, è molto semplice e realizza le due alternative seguenti. In mancanza di condizione di salto, incrementa di 1 l'indirizzo precedente e porta questo nuovo valore nel registro degli indirizzi. (La successiva microistruzione è, cioè, quella che, nella memoria di controllo, segue immediatamente la precedente). In presenza di condizione di salto, la rete incrementa l'indirizzo precedente del valore indicato nel campo apposito della microistruzione. La somma dei due valori fornisce l'indirizzo della successiva microistruzione.

5. Realizzazione fisica del Firmware

Come è già stato detto, la dotazione di microprogrammi del sistema è contenuta in una memoria generalmente del tipo *di sola lettura (Read Only Memory,* nella terminologia anglosassone, o più semplicemente ROM), in una memoria, cioè, in cui la scrittura è possibile una tantum, mentre la lettura può essere eseguita un numero illimitato di volte. L'impiego di tale tipo di memoria per immagazzinare il firmware è dettato da ragioni di costo, di velocità e di sicurezza di funzionamento.

Una memoria di sola lettura consiste, sostanzialmente, in un reticolo di conduttori ortogonali, ad ogni incrocio dei quali può essere collocato selettivamente un elemento di connessione (v. fig. 7). L'informazione è rappresentata, in termini binari, dalla presenza o assenza di connessione (un bit per incrocio).

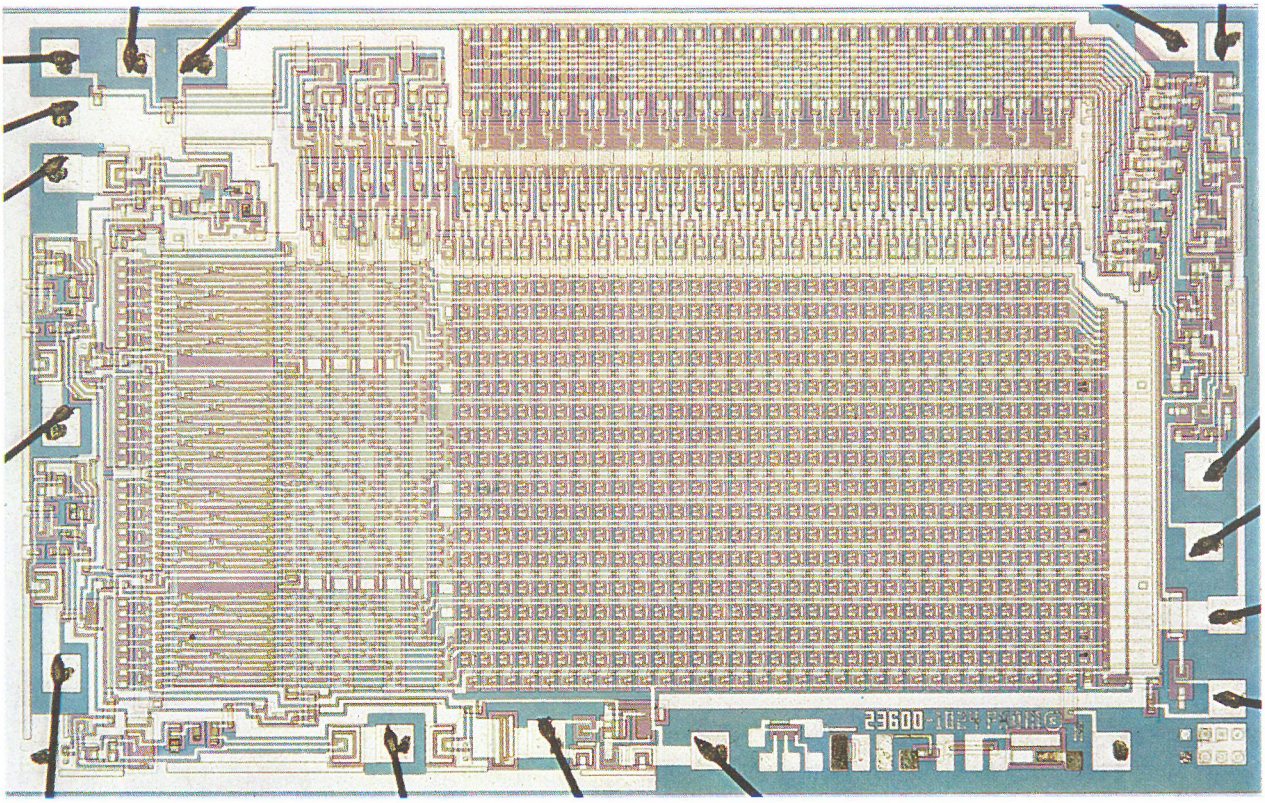
Applicando una eccitazione elettrica ad una linea orizzontale (ingresso), si ottiene un segnale su tutte le linee verticali (uscita) per le quali esiste connessione, e soltanto su queste.

Molte soluzioni tecnologiche sono state usate per realizzare questo tipo di memoria (ROM induttive, capacitive, resistive, a diodi, ecc.) [7]. Ma una soluzione veramente efficiente è arrivata soltanto in tempi recenti, con l'avvento dell'integrazione su larga scala (LSI), *(ROM a semiconduttore).* In questo caso l'elemento di connessione tra linea orizzontale e verticale è costituito da un transistore, e in un singolo « chip » possono essere contenuti migliaia di bit di memoria, insieme con tutti i circuiti di servizio (decodificatori, amplificatori), ecc. necessari per rendere funzionale l'insieme.

La soluzione accennata, anche se di gran lunga superiore ad ogni altra precedente, presenta tuttavia una rigidità di realizzazione, in quanto le connessioni selettive debbono essere fatte durante il processo di fabbricazione del « chip ». E' stata, pertanto, introdotta accanto alla ROM, la soluzione PROM *(Programmable Read-Only Memory),* ossia, Memorie di Sola Lettura Programmabili. In questa memoria esiste ad ogni incrocio un elemento di collegamento, in serie con un elemento alterabile elettricamente (ad esempio, un microfusibile). (fig. 8).

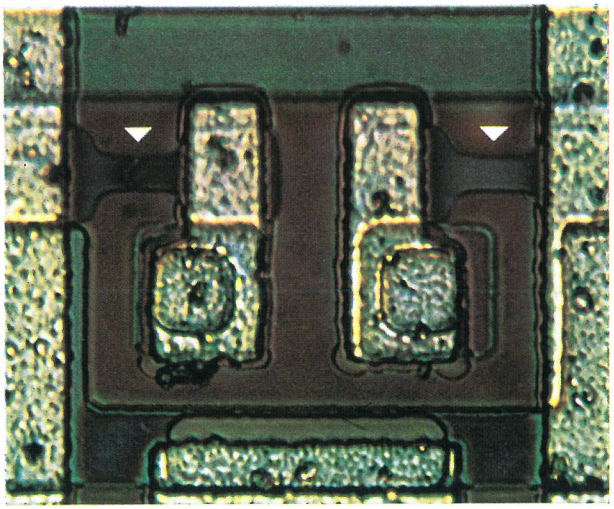
La connessione può essere mantenuta o interrotta a piacere, agendo sull'elemento alterabile. In pratica la « scrittura della memoria è realizzata applicando opportune tensioni ai terminali del dispositivo; questo processo è molto semplice e viene effettuato su dispositivi incapsulati, al momento dell'impiego.

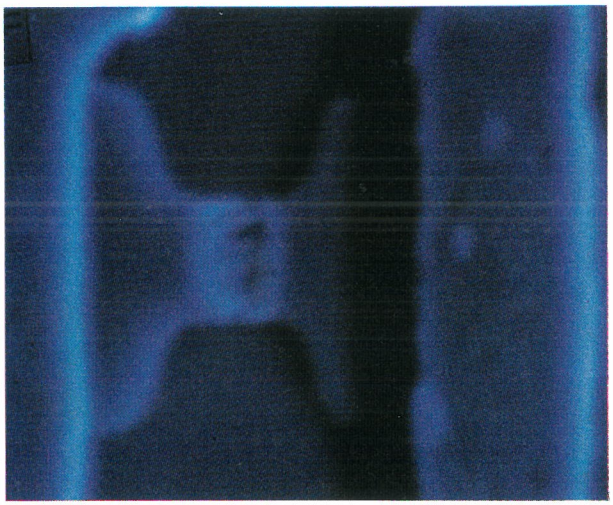
Le microfotografie che compaiono nella pagina 8 illustrano particolari delle memorie PROM impiegate in un elaboratore recentemente annunciato (« Sistema 62 »). In questo sistema, progettato a Pregnana dalla Honeywell Information Systems Italia, viene fatto un impiego sistematico ed estensivo del firmware.



Queste fotografie si riferiscono alla tecnologia di realizzazione del firmware di un avanzato sistema di elaborazione (« Honeywell 62 »). Le microfotografie illustrano il dispositivo integrato che realizza la memoria di sola lettura programmabile (PROM), *in cui il firmware del sistema risiede.*

La fotografia in alto mostra un « chip » (dimensioni reali circa 2,5x4 mm), in cui è contenuta una matrice di elementi del tipo illustrato in fig. 8, completa dei circuiti di servizio. L'elemento alterabile elettricamente è, in questo caso, un microfusibile, che può essere interrotto opzionalmente, inviando attraverso di esso un opportuno impulso di corrente. La seconda fotografia dall'alto rappresenta un dettaglio della precedente. Sono visibili due degli incroci della matrice, uno con fusibile intatto, l'altro con fusibile interrotto (in altre parole, un bit « 1 » ed un bit « 0 »). Nella figura in basso si vede invece, sotto ulteriore ingrandimento, un singolo fusibile interrotto.





(Le prime due fotografie sono state fatte al microscopio ottico e la terza a quello elettronico, presso il Centro di Ricerca e Progettazione di Pregnana Milanese della Honeywell Information Systems Italia, dove il sistema « 62 » è stato progettato).

6. Considerazioni sui sistemi microprogrammati

Da quanto si è detto, il concetto di microprogrammazione ha introdotto un livello intermedio nella organizzazione gerarchica dei sistemi di elaborazione: il livello firmware. A questa nuova entità è devoluto il compito di mediatore tra le due componenti classiche — il software e l'hardware — che in questa struttura non hanno più alcun canale di comunicazione diretta.

E' chiaro che questo nuovo elemento introduce ulteriori gradi di libertà della strutturazione dei sistemi di elaborazione. L'ottimizzazione in sede di progetto presenta perciò una problematica complessa, poiché ci si può muovere fra due estremi teorici che sono tra loro molto lontani, cioè: all'estremo inferiore, microistruzioni che pilotano pochissime linee di controllo (al limite una), e, all'estremo superiore, microistruzioni che coincidono con l'istruzione. E' da notare che questi limiti approssimano abbastanza bene i punti di progetto di calcolatori molto piccoli e molto grandi, dove per calcolatori molto piccoli si intendono macchine con tempi medi di esecuzione dell'istruzione dell'ordine del centinaio di microsecondi e per calcolatori molto grandi macchine con tempi medi di esecuzione dell'istruzione dell'ordine di microsecondi.

A prescindere dalla problematica presentata al progettista di sistemi di elaborazione da questa nuova entità, quali sono i vantaggi dal punto di vista dell'utente?

In generale, si può dire che il firmware ha reso attuali, nei sistemi, capacità e prestazioni non ottenibili con le tecnologie convenzionali se non a patto di inaccettabili aumenti dei costi.

Tre aree fondamental sono, in particolare, da mettere in evidenza a questo proposito, e cioè l'emulazione, l'affidabilità dei risultati e l'autodiagnosi.

Per *emulazione* si intende la capacità di un sistema di eseguire programmi scritti nel linguaggio macchina di altri sistemi. In particolare, questa capacità permette all'utente di trasferire inalterato tutto il suo patrimonio di programmi, nel passaggio da un sistema vecchio ad uno nuovo. E' chiaro che la realizzazione di questa funzione comporta la realizzazione di altre macchine virtuali sulla stessa macchina ospite. Per quanto è stato detto, la soluzione firmware, vero e propiro « adattatore » tra software e harware, risulta la soluzione naturale di questo problema. Occorre anche ricordare che nel concetto di emulazione è basata la possibilità di realizzare *famiglie di calcolatori*, delegando al firmware il compito di rendere perfettamente compatibili dal punto di vista linguaggio di macchina sistemi nettamente diversi sotto l'aspetto hardware.

Garantire l'*affidabilità* dei risultati delle elaborazioni è un requisito fondamentale. Questo problema ha due livelli di complessità. Il primo livello è la rilevazione dell'errore hardware o software, il secondo è l'inibizione della propagazione degli effetti dell'errore. Questo secondo aspetto è fondamentale quando si fa uso della multiprogrammazione; infatti è evidente che sarebbe disastroso se un errore, avvenuto ed individuato in una certa area del sistema, andasse a deteriorare un'altra area in modo tale da comprometterne la integrità dei dati senza che ce se ne possa accorgere. Al fine di evitare questi gravi inconvenienti, occorre eseguire una serie di controlli del sistema durante il suo funzionamento, a livello sia software che hardware. Per quanto riguarda il software, i compiti svolti dal firmware vanno dal controllo della validità formale di istruzioni e dati, alla verifica che l'accesso della memoria espresso dalla istruzione in esecuzione sia corretto, al controllo che l'esecuzione della istruzione sia nei diritti del programma che la richiede. Per l'hardware, i controlli da firmware consistono nel riconoscimento del buon funzionamento di reti logiche e memorie.

Per *autodiagnosi* si intende la capacità della macchina di scoprire e segnalare i propri organi guasti. Il procedimento di autodiagnosi consiste, concettualmente, nel sottoporre la rete in esame ad una serie di stimoli e nel controllare che la risposta sia corretta. E' chiaro che l'approccio firmware, data la visibilità, quasi a livello « gate », che esso offre, garantisce una capacità di isolamento dell'eventuale dispositivo guasto con una risoluzione che non sarebbe altrimenti possibile raggiungere. Tutto ciò si traduce, alla fine, in un importante passo avanti per quanto riguarda i costi e i tempi di manutenzione.

Come ultima osservazione si può anche dire che l'impiego del firmware ha reso possibile l'inserimento in sistemi medio-piccoli di prestazioni e funzionalità in precedenza riservate, per ragioni di costo, ai grandi sistemi di elaborazione. L'inserimento di nuove funzionalità ha portato, come conseguenza, ad una evoluzione dell'architettura di queste macchine verso strutture molto generalizzate, dove più che la singola funzionalità si tende ad ottimizzare il sistema nel suo complesso al fine di soddisfare tutte le funzionalità richieste nel modo più armonico possibile.

7. Conclusione

I sistemi microprogrammati rappresentano il più avanzato « stato dell'arte nel campo dell'elaborazione dei dati. Ma le possibilità di progresso in questo ambito sono tutt'altro che esaurite. E' anzi probabile che, in futuro, il firmware tenda sempre più ad invadere quei campi che, tuttora, sono di stretto dominio del software. E' questa la direzione di numerose ricerche attualmente in corso in parecchie parti del mondo, presso università e industrie.

Uno dei campi di maggior interesse è quello della realizzazione tramite firmware di interpretatori di linguaggi ad alto livello. Lo scopo che ci si propone è la eliminazione del compilatore (ossia del programma di sistema che trasforma il programma in

linguaggio ad alto livello in un altro in linguaggio macchina) o comunque una riduzione della sua complessità. Ciò porterebbe ad una maggiore efficienza dei sistemi in termini di tempi globali di esecuzione nonché ad una facilitazione nella produzione del software.

Un'altra direzione è la realizzazione in firmware di parte dei sistemi operativi. Anche questa prospettiva è del massimo interesse, data la enorme complessità raggiunta da questi ultimi.

Un limite pratico alle evoluzioni accennate è costituito, attualmente, dal costo delle memorie in cui il firmware risiede. Ma il continuo progresso della tecnologia elettronica lascia prevedere soluzioni adeguate per un ulteriore, intensivo, impiego del firmware.

Bibliografia

[1] A. OPLER, *Fourth-Generation Software,* Datamation, vol. 13, n. 1, Gennaio 1967.

[2] M.W. WILKES, *The best way to design an automatic calculating machine,* Manchester University Computer Inaugural Conference, 1951.

[3] R.J. MERCER, *Microprogramming,* Journal of the Association for Computing Machinery 4, n. 2, 1957.

[4] M.W. WILKES, *The growth of interest in microprogramming: a literature survey,* Computer Surveys 1, n. 3, 1969.

[5] R.F. ROSIN, *Contemporary concepts of microprogramming and emulation,* Computing Surveys 1, n. 4, 1969.

[6] S.S. HUSSON, *Microprogramming: principles and practice,* Prentice Hall, 1970.

[7] IEEE TRANSACTION ON COMPUTERS, Vol. C-20, n. 7, Luglio 1971. Completamente dedicato alla microprogrammazione.

[8] R. LASCHI, G. NERI, *Tecniche di microprogrammazione,* Alta Frequenza, Vol. 42, n. 9, Settembre 1973.

[9] F. FILIPPAZZI, *Sistemi di memoria con lettura non distruttiva,* Alta Frequenza, Vol. 33, n. 7, luglio 1969.

La correttezza dei programmi

GIOVANNI DEGLI ANTONI
Università degli Studi di Milano
Gruppo di Elettronica e Cibernetica

1. Introduzione

Ogni innovazione tecnologica determina ed è determinata da nuovi orientamenti nel settore della ricerca pura ed applicata.

A questa affermazione schematica non sfugge neppure lo sforzo che i ricercatori di molti laboratori stanno facendo per controllare alcuni aspetti pratici e teorici relativi alla programmazione di elaboratori elettronici.

La importanza pratica di questi ultimi e la coscienza che il software debba considerarsi come un nuovo prodotto industriale e non una attività artigianale, ha reso talvolta notevoli gli sforzi diretti alla elaborazione di teorie relative alla attività di produzione del software stesso. Complessivamente questi sforzi sono diretti alla messa a punto di metodi che permettano una produzione più accurata di programmi facilitando il compito del programmatore che potrà così dedicarsi agli aspetti più creativi della attività di programmazione.

D'altra parte la dimensione dei prodotti software attuali (anche diversi milioni di linee di codice) assieme alla coscienza della impossibilità di realizzarli privi di errori, ha orientato la ricerca a risolvere in qualche modo il problema della « correttezza dei programmi » (1). Ma, come spesso accade nelle scienze applicate, la possibilità di raggiungere obiettivi anche di carattere meramente applicativo passa attraverso la costruzione di fondamenti che sfiorano gli aspetti più profondi della conoscenza. Nel caso del problema della correttezza dei programmi, tale situazione si è presentata di nuovo fornendo al ricercatore l'occasione di costruire una solida teoria dei programmi che dovrebbe permettere di raggiungere gli obiettivi proposti, ovvero la meccanizzazione di parte della attività legata alla costruzione dei programmi.

In queste note cercheremo di delineare alcune tappe e le linee di sviluppo di questa attività rivolta ad assicurare la correttezza dei programmi, fornendo anche alcune indicazioni sulla validità pratica della via seguita.

(1) A titolo indicativo dell'importanza di questo problema, secondo stime americane la ricerca e la correzione degli errori di un programma rappresentano circa il 40% dello sforzo complessivo di realizzazione del programma stesso. [1]

2. Il problema generale della correttezza

Che cos'è un programma? La domanda non è priva di interesse. Infatti non si potrà risolvere il problema della correttezza dei programmi senza affrontare il problema di definirli.

Visti dall'esterno, i programmi possono essere pensati come una sequenza di fori in un qualche pacco di schede, ovvero una sequenza di simboli su un tabulato opportuno, o anche una frase scritta in un qualche linguaggio; brevemente, potremo pensarli come una sequenza di segni formata con certe convenzioni: le regole del linguaggio. Tale linguaggio sarà orientato a compiti specifici e non costituisce patrimonio di una certa popolazione in un certo luogo; non potrà pertanto essere definito come un linguaggio naturale. Tuttavia dovrà essere pensato come un linguaggio artificiale destinato alla comunicazione umana. Esattamente come i linguaggi naturali, anche i linguaggi artificiali si prestano alla costruzione di infinite frasi (i programmi) e saranno analizzabili in termini *lessicali, grammaticali* e *semantici.* Il lessico e la grammatica indicheranno le regole di formazione delle frasi del linguaggio di programmazione scelto. La semantica ne indicherà il significato.

Lo studio grammaticale (o sintattico) dei linguaggi di programmazione è molto importante ed è stato certamente il primo passo per la realizzazione di metodologie in grado di contribuire alla costruzione di programmi corretti.

Infatti la possibilità di disporre di regole di for-

mazione accurate (la grammatica o sintassi) permette di meccanizzare la verifica della corretta costruzione dei programmi stessi rispetto alle regole date. Ciò avviene con tutti i moderni linguaggi di programmazione durante la cosiddetta fase di « compilazione ». In questa fase, che precede l'esecuzione vera e propria da parte di un elaboratore elettronico, vengono dapprima controllati i programmi per stabilire se sono grammaticalmente corretti.
Qualora ciò non si verifichi, l'elaboratore indicherà gli errori che il programmatore ha commesso mediante opportuni messaggi diagnostici o addirittura correggerà automaticamente certi errori.
Quest'ultima situazione sarà possibile solo in quei casi in cui la sintassi del linguaggio di programmazione imporrà al programmatore di esprimere le sue intenzioni più di una volta, in modo che un opportuno meccanismo sia in grado di individuare e correggere gli errori. La fase di compilazione termina allorché il programma originale è tradotto in una forma più adatta alla diretta esecuzione.
Gli aspetti grammaticali sono certamente molto importanti, ma trascurano la possibilità di tenere conto del significato di un programma. Questo potrà essere compreso osservando innanzi tutto che i programmi sono destinati ad evocare sequenze di esecuzione da parte di un elaboratore elettronico. L'elaboratore elettronico è in questa luce un complicato automa che esegue solamente ordini.
Ma è solo questo il significato di un programma? Certamente dal punto di vista di un elaboratore ciò è corretto. Ma dal punto di vista umano la cosa è più complessa. Infatti l'uomo attribuisce un significato alle sequenze di esecuzione effettuate, non in quanto tali ma in quanto originate da una motivazione: la motivazione del programma. E' stata la motivazione che il programmatore aveva in mente a suggerire la costruzione di un programma che evocasse sequenze di esecuzione a cui si può attribuire un significato: il significato del programma.
Dunque i termini in gioco sono:
a) Motivazione di un programma.
b) Significato di un programma.
Mediante questi termini, di cui daremo nel seguito una qualche definizione, è possibile porre il problema generale della correttezza in uno dei seguenti modi:

a) Approccio analitico (problema di analisi).
Verificare se il significato (della esecuzione) di un programma assegnato corrisponde alla sua motivazione.

b) Approccio sintetico (problema di sintesi).
Assegnata una motivazione costruire un programma il cui significato (il significato della cui esecuzione) corrisponda alla motivazione assegnata. Definire, cioè, una metodologia costruttiva dei programmi che ne assicuri la correttezza.

3. Il significato di un programma

Per comprendere il significato della esecuzione di un programma si osservi che un elaboratore, sotto la azione di un programma, effettua una sequenza deterministica (ripetibile, prevedibile) di operazioni. Ciò ha come conseguenza che ad una configurazione della memoria dell'elaboratore prima della esecuzione del programma corrisponderà una ed una sola configurazione della memoria stessa dopo la esecuzione del programma. Se immaginiamo le configurazioni come punti dello spazio dei dati, potremo affermare che un programma computa una funzione ovvero una corrispondenza univoca fra le configurazioni di ingresso e quelle di uscita [2]. A titolo di esempio consideriamo il seguente programma:

begin z : = 0; u : = x;
repeat z : = z + y; u : = u — 1
until u = 0
end

I vari segni che compaiono verranno interpretati dall'elaboratore nel modo seguente:

Inizia la esecuzione **(begin)** *registrando la configurazione chiamata 0 nella cella di memoria chiamata z (z : = 0) e quindi registra il contenuto della cella chiamata x nella cella chiamata u. Ciò fatto, ripeti i seguenti comandi finché il contenuto della cella u è la configurazione chiamata 0: registra nella cella chiamata z il risultato della azione chiamata + effettuata sul contenuto della cella chiamata z e sulla cella chiamata y; sostituisci inoltre il contenuto della cella chiamata u con il risultato della azione chiamata — effettuato sul contenuto della cella u stessa e sulla configurazione chiamata 1 (***repeat,** *ecc.); termina ora la esecuzione* **(end)**.

E' semplice interpretare in senso matematico alcuni dei segni che compaiono se immaginiamo che il programma operi su quelle entità chiamate numeri interi. In tal caso, ad esempio, il simbolo + indicherà la somma fra interi; il simbolo — la differenza; il simbolo 0 lo zero. Gli altri segni del programma potranno essere interpretati in modo da fornire un significato complessivo matematico a tutto il programma. Nel caso del programma ora visto è facile rendersi conto che la funzione computata è il prodotto. Ovvero indicati con X, Y due arbitrari interi positivi, il programma fornirà, una volta eseguito, il numero Z dato da:

$$Z = X * Y$$

Dunque la funzione prodotto è il significato del programma considerato. Il programma specificherà dunque un modo per computare la funzione, mentre il suo significato sarà una corrispondenza nello spazio dei dati. Quest'ultima discussione ha voluto sottolineare che gli aspetti algoritmici con cui viene raggiunto un certo risultato sono descritti dal programma, mentre il suo significato è proprio il risultato raggiunto. In breve: un programma descrive *come*

[2] Poiché il programma può non terminare per certi valori di ingresso, in generale la funzione che questi computa è parziale, ovvero non è sempre definita.

si computa un certo risultato; il suo significato *cosa* si computa.
La attribuzione del significato di un generico programma in termini di funzioni note è un arduo problema che non sempre è risolubile senza introdurre sulle funzioni note operazioni opportune. In termini astratti il problema di attribuire un significato matematico ad un programma è stato risolto da Scott [2] nei termini della cosiddetta teoria del punto fisso.
Secondo questa teoria, un programma ha come significato una particolare funzione che può essere determinata come la soluzione minima (in un senso opportuno) di una equazione funzionale associabile al programma considerato. In tale approccio la costruzione della equazione funzionale da associare al programma è piuttosto semplice e può avvenire in sostanza per un processo di traduzione del programma assegnato. La soluzione della equazione, ovvero la determinazione del significato del programma, richiede inventiva matematica qualora sia richiesta in modo esplicito. Qualora sia richiesta solamente una soluzione simbolica, questa è immediatamente disponibile utilizzando un opportuno operatore. La importanza di questa soluzione simbolica è dovuta al fatto che gode di tutte le proprietà della funzione computata dal programma in modo che si possono effettuare studi sulla funzione in esame quali il confronto con altre funzioni, determinazione di proprietà, eccetera. E tutto ciò può essere effettuato con strutture di dati arbitrarie (numeri, liste, array, eccetera).

4. La motivazione di un programma

La possibilità della soluzione del problema della correttezza richiede la conoscenza della motivazione dei programmi. Questa necessità è il limite pratico alla applicazione diretta dei metodi della correttezza ai casi reali. Infatti spesso la descrizione del programma o non esiste o è del tutto frammentaria.
Il programma costituisce in questi casi l'unico documento accurato disponibile e pertanto sarà impossibile, per la mancanza di informazione sulla motivazione, stabilirne la correttezza.
Tale situazione ha due origini differenti. La prima è legata al fatto che i metodi attualmente disponibili per la valutazione della correttezza e/o per la sintesi vengono raramente impiegati e quindi le motivazioni non sono documentate. La seconda è legata al fatto che talvolta la motivazione di un programma è il programma stesso ovvero la sequenza degli eventi che questi determina e non la funzione che il programma computa.
La possibilità di esprimere la motivazione di un programma è legata alla disponibilità di un linguaggio (non di programmazione) in cui tale motivazione sia esprimibile. In mancanza di tale linguaggio le motivazioni vengono espresse nel linguaggio naturale eventualmente con l'ausilio di notazioni a cui si possa attribuire un qualche significato matematico.

I seguenti sono esempi di motivazioni:

1. Calcolare il prodotto di due numeri interi positivi o nulli.

2. Per ogni coppia di interi positivi o nulli X, Y, calcolare la funzione $Z = X*Y$.

3. Invertire una lista.

4. Tradurre un qualsiasi programma ALGOL in un programma FORTRAN che computa la medesima funzione.

5. Emulare la istruzione della macchina X con un microprogramma della macchina Y.

6. Determinare l'unico valore di z, se esiste, che per qualsiasi coppia di interi positivi o nulli x, y renda soddisfatta la seguente eguaglianza:

$$x + z = y$$

7. Calcolare il massimo comun divisore di due interi assegnati positivi arbitrari.

Discutiamo brevemente gli esempi di motivazione. Il primo, espresso in un linguaggio naturale, esprime come richiesta di significato di un programma la funzione prodotto. Tale identica motivazione è espressa in un modo più formale nel secondo esempio, ottenibile dal primo. Questi è anche un esempio di specificazione di un programma attraverso una relazione entrata/uscita e la delimitazione dei valori di ingresso. La relazione entrata/uscita sarà espressa mediante una formula che risulterà soddisfatta da coppie di valori e/u tali che, se e costituisce l'ingresso al programma espresso dalla motivazione, u risulterà la uscita dello stesso, ovvero il valore computato. Il terzo esempio risulta alquanto più complicato da esprimere nonostante la apparente semplicità. La funzione che il programma relativo a questo esempio computerà sarà indicata con « reverse(x) », essendo x una generica lista da invertire.
La motivazione 4 è alquanto chiara a chi sappia cosa significhi programma ALGOL e programma FORTRAN. Una considerazione analoga vale per la motivazione successiva che fa riferimento a concetti di microprogrammazione. Queste ultime motivazioni potrebbero essere espresse in forma simbolica, ma probabilmente la espressività, in termini umani, ne verrebbe alquanto diminuita.
La motivazione 6 è alquanto interessante in quanto non è esplicitato il significato della funzione che il programma associato a quella motivazione è destinato a computare. In effetti la funzione significato implicita è la funzione (parziale) differenza fra y ed x. L'ultimo esempio infine fornisce il nome della funzione che il programma dovrà computare senza specificarne la natura. Questa, diversamente dalla motivazione 6 che contiene tutta la informazione sulla funzione da computare, cioè sulla differenza, dovrà essere desunta da conoscenze di matematica originate dalla definizione di massimo comun divisore.
Tutte queste motivazioni, che appaiono anche molto differenti, costituiscono solo esempi di frasi di linguaggi: i linguaggi delle motivazioni. Tali linguaggi

variano dal linguaggio naturale ad un linguaggio della logica in cui si possano esprimere relazioni su certi enti (gli interi, le liste, i programmi, eccetera). Quanto più accurata è la definizione di un linguaggio in cui si esprimono le motivazioni, tanto maggore sarà la possibilità di utilizzare procedure meccanizzabili quale ausilio allo studio della correttezza dei programmi. La letteratura corrente è alquanto carente relativamente agli aspetti dello studio delle caratteristiche di un linguaggio adatto ad esprimere motivazioni di programmi. In ogni caso un linguaggio in cui si esprimono motivazioni avrà caratteristiche formali analoghe a quelle di un linguaggio di programmazione: esisterà una grammatica, le sue frasi (le motivazioni) esprimeranno funzioni (senza tuttavia specificare come queste potranno venir computate, ovvero, interpretate), definiranno oggetti matematici che altro non sono che funzioni.
La differenza essenziale fra motivazione e programma sarà pertanto dovuta al fatto che la prima non suggerirà direttamente alcun procedimento di computazione che permetta di calcolare i valori delle funzioni espresse per certi valori di ingresso, mentre ciò è carateristico del programma.

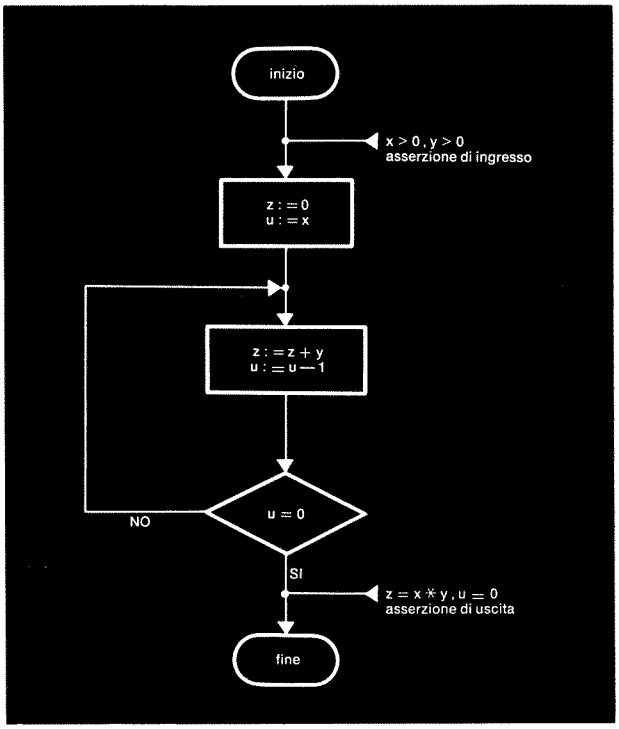


Fig. 1 - Diagramma di flusso di un semplice programma (prodotto di x e y).

5. La correttezza dei programmi: approccio analitico

La discussione fatta sul significato dei programmi e sulla loro motivazione, suggeriscono direttamente vari metodi per la dimostrazione della loro correttezza. Tutti i metodi dovranno verificare matematicamente che la motivazione del programma corrisponde al significato dello stesso. E ciò può essere fatto in vari modi. Si potrebbe ad esempio cercare una qualche motivazione compatibile con il programma assegnato e confrontare la nuova motivazione con quella data. Un approccio del genere avrebbe il vantaggio di trasformare il problema della correttezza, una volta determinata la nuova motivazione, in un problema squisitamente di logica matematica. Ciò è possibile grazie alla teoria del punto fisso.
Come abbiamo detto, secondo questa teoria si costruisce una motivazione (equazione funzionale) mediante un procedimento di traduzione a partire dal programma assegnato. La motivazione trovata dovrà essere confrontata in modo opportuno con la motivazione assegnata.
Il procedimento si presta anche in casi in cui la motivazione è espressa in un linguaggio naturale se si conoscono alcune proprietà della funzione computata dal programma che la caratterizzano completamente.
Per illustrare quest'ultimo caso, si supponga di disporre di un programma, che indicheremo con PROG, destinato ad invertire una lista (esempio di motivazione 3). Mediante la semantica matematica di Scott, determiniamo la funzione computata dal programma che chiamiamo « reverse ». « Reverse » sarà costituita da una espressione simbolica che dipenderà dal programma PROG in esame. Per dimostrarne la correttezza ci basterà dimostrare che:

$$reverse\,[reverse\,(x)\,] = x$$

quale che sia x.
Un altro approccio, che è alquanto più impiegato nelle applicazioni, e che permette il confronto fra motivazioni e programmi è dovuto a Floyd [3]. Le motivazioni saranno espresse come limitazioni dei dati di ingresso e relazione entrata/uscita. Le limitazioni dei dati di ingresso e la relazione entrata/uscita nell'approccio di Floyd sono considerate asserzioni da associare rispettivamente alla entrata ed alla uscita del programma. Si associano al programma altre opportune asserzioni in punti interni allo stesso. Si cercherà di dimostrare che il programma e tutte le asserzioni sono compatibili. La difficoltà dell'approccio è legata alla difficoltà di costruire le asserzioni dei punti interni al programma che esprimano l'effetto della esecuzione del programma per ogni possibile computazione. In effetti tali asserzioni possono essere costruire solo da chi ha una profonda conoscenza del programma in esame.
Illustreremo il metodo di Floyd (detto anche delle asserzioni induttive) con un esempio.
Si consideri, per ragione di semplicità, il programma del paragrafo 2 sotto forma di diagramma di flusso (fig. 1). I segni che compaiono in questo saranno interpretati sugli interi.
Per tutte le coppie x, y di interi positivi, il programma fornirà il valore $z = x*y$. Tutto ciò può essere visualizzato associando le condizioni $z > O$, $y > O$ alla entrata del programma (asserzione di ingresso) e la condizione $z = x*y$ (asserzione di uscita) alla sua uscita come indicato nella stessa figura 1.

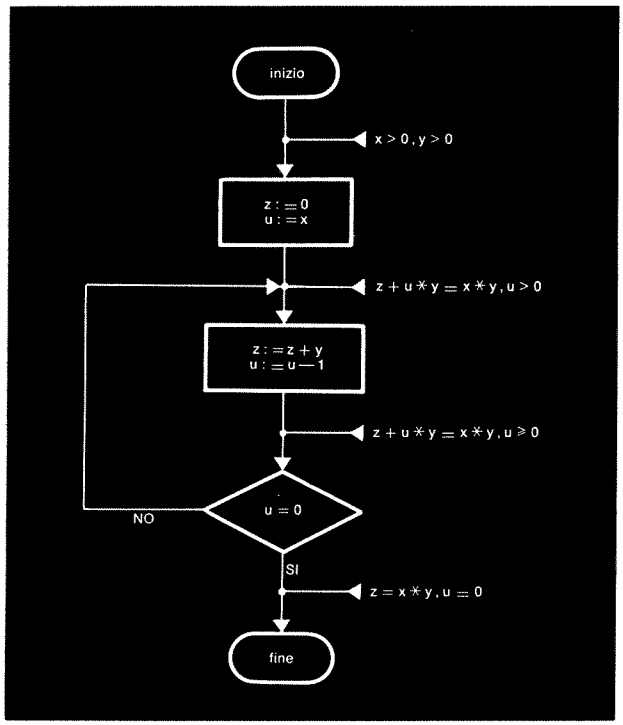


Fig. 2 - Inserimento di asserzioni interne per la verifica della correttezza del programma.

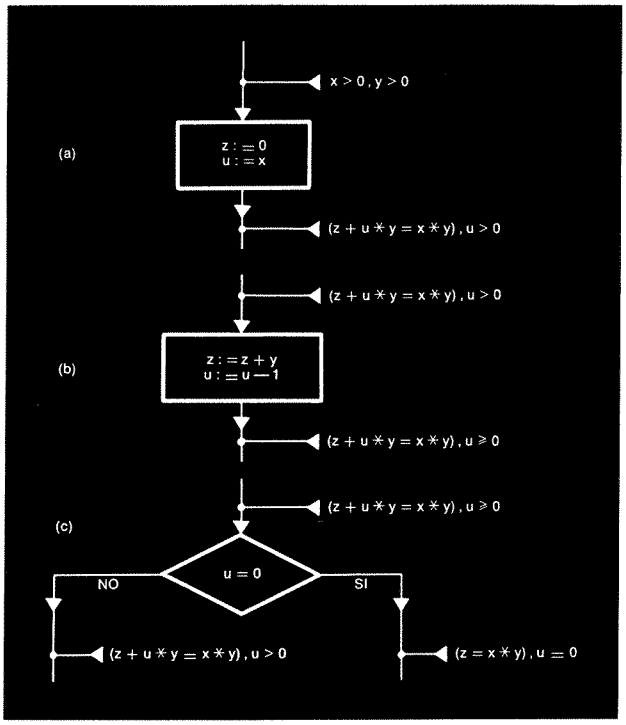


Fig. 3 - Frammentazione del programma per la verifica della correttezza.

Il programma risulterà corretto rispetto alle due condizioni se per tutte le coppie che soddisfano la asserzione di ingresso, qualora il programma termini, sarà soddisfatta anche la asserzione di uscita.

La dimostrazione della precedente affermazione, per essere effettuata, richiede la capacità di esprimere la trasformazione effettuata dal programma (e non quella che vogliamo il programma compia) sui dati di ingresso. Per evitare ciò, si possono introdurre opportune asserzioni nei punti interni al programma che esprimono il legame fra le variabili di programma nei punti a cui le asserzioni sono associate, ogni qualvolta il controllo venga trasferito ai punti considerati.
Nel caso del programma in esame le asserzioni interne da introdurre sono rappresentate in figura 2. Le asserzioni, come abbiamo già detto, debbono essere inventate dal programmatore che ha costruito il programma o comunque da chi ha una profonda conoscenza dello stesso poco prestandosi (se pure si prestano) alla costruzione per via meccanica.
Assegnate le asserzioni, la dimostrazione della correttezza si otterrà dimostrando che:

Le asserzioni agli estremi di un qualsiasi percorso sono compatibili con la trasformazione effettuata dal percorso stesso.

I percorsi da analizzare sono facilmente individuabili spezzando il programma in frammenti, come suggerito da Floyd, che terminino con asserzioni e non comprendano loops o cicli.
Il risultato di questa frammentazione nel caso dell'esempio in esame è mostrato in figura 3. Si badi che la frammentazione non richiede inventiva matematica.

A questo punto il lettore non avrà grande difficoltà a convincersi che tutti i frammenti di programma considerati sono compatibili con le asserzioni agli estremi e quindi il programma che ha dato origine ai frammenti risulterà corretto. Per dimostrare queste ultime affermazioni procediamo a partire dal caso (a) della figura 3.
Supponiamo che prima della esecuzione del frammento (a) si disponga di una coppia di valori x, y che soddisfano la asserzione posta all'ingresso del frammento (a) stesso:

$$x > 0,\ y > 0$$

e valori arbitrari di z ed u. Dopo la esecuzione del frammento (a), la asserzione di uscita sarà verificata? Per verificare ciò basta sostituire *nella asserzione di uscita* al posto di z il suo nuovo valore 0 ed al posto di u il suo nuovo valore x; si ottiene:

$$0 + x*y = x*y,\ x > 0$$

ovvero:

$$x*y = x*y,\ x > 0$$

Questa asserzione è certamente vera se la asserzione di ingresso è verificata. Quindi, nel primo frammento, asserzioni e trasformazioni sono compatibili.
Analizziamo in modo analogo il secondo frammento (b).
Ancora supporremo di disporre di valori che soddisfano la relazione di entrata (di questo frammento):

$$z + u*y = x*y,\ u > 0$$

La trasformazione effettuata dal frammento in esame è la seguente:

z viene sostituito con $z + y$
u viene sostituito con $u - 1$

se sostituiamo questi nuovi valori ancora nella *asserzione di uscita* del frammento in esame, otteniamo:

$$(z + y) + (u - 1) * y = x * y,\ u - 1 \geqslant 0$$

che è chiaramente soddisfatta se la asserzione di ingresso è soddisfatta. Lasciamo al lettore la verifica del caso (c), che del resto è pressoché immediata poiché la prova non altera il valore delle variabili di programma. Tutti i frammenti sono dunque compatibili con le relative asserzioni. Poiché in queste sono comprese anche le asserzioni di entrata e di uscita, il programma risulterà corretto anche rispetto a queste, ovvero qualora il programma termini, se la asserzione di entrata risulterà soddisfatta, risulterà soddisfatta anche la asserzione di uscita.
I punti salienti del procedimento seguito sono:

— Al programma vengono associate delle asserzioni che specificano i domini dei dati di ingresso e la relazione ingresso/uscita.

— A punti interni del programma vengono associate asserzioni. Occorrerà almeno una asserzione per ogni « loop » del programma.

— Si frammenta il programma in cammini (senza cicli o loops) ai cui estremi sono associate asserzioni.

— Si verifica che tutti i cammini ottenuti dalla precedente frammentazione sono compatibili con le proprie asserzioni.

Se l'ultimo punto sarà verificato il programma risulterà verificato rispetto alle condizioni di entrata e di uscita assegnate.
Si badi però che la tecnica adottata di frammentazione dei percorsi non garantisce che il programma termini per tutti i valori per cui il programma risulta corretto. La terminazione dovrà pertanto essere stabilita con altri metodi, originati sempre da Floyd.

6. La correttezza dei programmi: approccio sintetico

Daremo qui brevemente solo alcuni cenni sull'approccio sintetico. Questi si basa sul fatto che se disponiamo della motivazione e del programma, debbono essere svolti due lavori creativi per dimostrarne la correttezza. Dapprima viene infatti costruito il programma. Su di questi, con uno sforzo che richiede una notevole creatività, quale ad esempio la costruzione delle asserzioni, si procede ad una non sempre semplicissima verifica della correttezza.
Nell'approccio sintetico il procedimento, pur richiedendo inventiva matematica è diretto: si procederà dalla motivazione del programma con un procedimento possibilmente formale e di cui sarà facile fornire garanzie di correttezza. L'approccio sintetico, altrimenti definito anche « costruttivo », è sostanzialmente la base della cosiddetta *« programmazione strutturata »*.
Seguendo Dijkstra [4], che ne è il principale autore, nella programmazione strutturata l'approccio costruttivo è limitato ad un livello di formalismo piuttosto modesto, cosicché risulta impossibile adottare una qualsivoglia forma di verifica del procedimento di costruzione che faccia ricorso all'impiego di procedure meccanizzate.
L'approccio sintetico è oggetto attualmente di ricerca. Le basi matematiche dell'approccio risiedono nella cosiddetta matematica costruttiva che, peraltro, si propone di mostrare come si computino oggetti di cui si dimostra la esistenza ([3]).

7. Discussione

Attualmente, nella quasi totalità dei casi, la presunzione che un programma è corretto è basata sui test che questi subisce durante o dopo il suo sviluppo. Questi test sfortunatamente non garantiscono la correttezza del programma per tutti i possibili dati di ingresso ma solo per quei dati per cui il programma è stato verificato. Tuttavia i test hanno un valore ben superiore a quello appena dichiarato poiché costituiscono una verifica seppure parziale, dei ragionamenti che hanno condotto alla costruzione del programma.
La correttezza dei programmi, sia nel suo approccio analitico che in quello sintetico, permetterebbe di superare le difficoltà indicate poiché avrebbe valore per tutti i dati di ingresso ad un programma, qualora fosse praticamente attuabile.
Le esperienze effettuate, anche in ambiente industriale [6], hanno dimostrato che attualmente l'approccio analitico ha valore pratico anche in porzioni già notevoli di programmi. Tuttavia valutazioni del costo non sono disponibili, mentre risulta che le difficoltà connesse con le dimostrazioni matematiche hanno scoraggiato una adozione diffusa anche a causa della carenza dei metodi automatici di dimostrazione di teoremi.
La possibilità di meccanizzare parti della attività di dimostrazione della correttezza è stata dapprima considerata da King [7] sulla base dell'approccio di Floyd. King ha realizzato un programma che accetta come ingressi il testo del programma e le asserzioni. Il programma tenta di dimostrare la correttezza sulla base delle informazioni indicate. Se il tentativo ha successo viene fornita la prova della correttezza. In caso di insuccesso il programma richiede indicazioni sul proseguimento della esecuzione o, eventualmente, fornisce controesempi se risulta che il programma in esame è errato.
La possibilità di impiegare un elaboratore nell'ausilio o alla dimostrazione della correttezza o della sintesi dei programmi si ha solamente nei casi in cui si disponga di un linguaggio formale per indicare le asserzioni o le motivazioni e naturalmente i programmi. Tale linguaggio formale, a parte il già citato caso di King, è stato particolarmente curato da Milner [8] che si è basato sulla teoria del punto fisso per la formalizzazione dei concetti interessati.

([3]) Una prima versione di un procedimento di costruzione di programmi con metodi formali a partire dalla motivazione espressa in un linguaggio della logica è stata proposta da Degli Antoni, Miglioli, Ornaghi. [5]

In tutti i casi comunque, la presenza di un linguaggio formale ha come principale vantaggio la possibilità di meccanizzare la *verifica* che la dimostrazione di correttezza sia stata effettuata conformemente alle regole formali in gioco, e non tanto l'ausilio nella dimostrazione vera e propria che sarà sostanzialmente a cura di colui che costruisce il programma.
Quindi sebbene sia ancora lontano il tempo in cui si utilizzeranno metodi di dimostrazione automatica, non è tuttavia al di là delle possibilità attuali verificare il lavoro di chi si sforza di costruire programmi corretti con metodi sia analitici che sintetici. Nel caso di metodi sintetici la verifica della correttezza attraverso la verifica del procedimento di costruzione si presenta particolarmente attraente. Ben inteso a quel punto apparirà in tutta la sua luce il problema di convalidare le motivazioni: problema attualmente offuscato dalla difficoltà di utilizzare tali motivazioni nei casi in cui siano note.

Bibliografia

[1] A. CICU, *Problemi della produzione industriale di software*, Rivista di Informatica, Vol. 4, n. 3-4, 1973.

[2] D. SCOTT, *Lattice theory, data thypes and semantics*, Formal Semantics of Programming Languages, Prentice Hall, 1972.

[3] R.W. FLOYD, *Assigning meaning to programs*, Proceedings of Symposium on Applied Mathematics, American Mahtematical Society, Vol. 19, 1967.

[4] E.W. DIJKSTRA, *A constructive approach to the problem of program correcteness*, B.I.T., Vol. 8, 1966.

[5] G. DEGLI ANTONI, P.A. MIGLIOLI, M. ORNAGHI, *Top-down approach to the synthesis of programs*, International Symposium on programming, Paris 1974.

[6] E. LANZARONE, M. ORNAGHI, *Un esempio di correttezza di un programma*, Rapporto interno Honeywell Information Systems Italia ed Università di Milano, 1973.

[7] J. KING, *A verifying compiler*, Debugging techniques of large systems, Prentice Hall, 1971.

[8] R. MILNER, R. WEYHRAUCH, *Program semantics and correctness in a mechanized logic*, First USA-Japan Computer Conference, 1972.

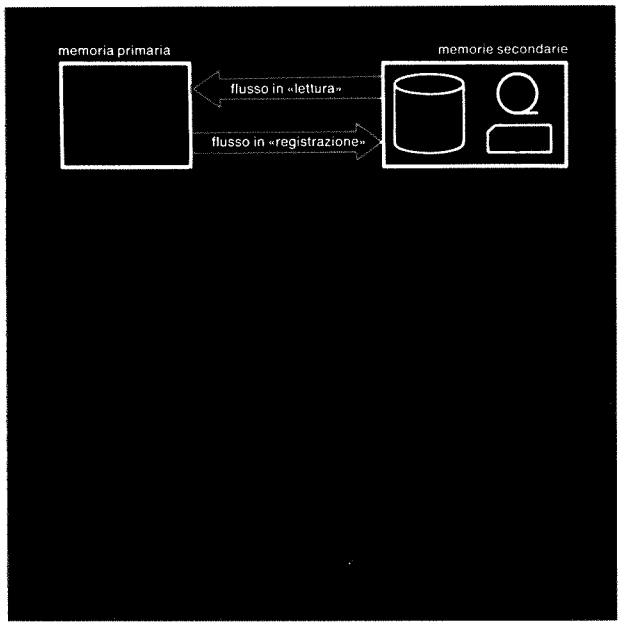
Il concetto di database

EMILIO RAJA
Honeywell Information Systems Italia
Centro di Ricerca e Progettazione di Pregnana Milanese

1. Introduzione

L'organizzazione dell'archivio dei dati costituisce un problema centrale nella utilizzazione di un sistema di elaborazione elettronica.
In questo settore, il « *database* » rappresenta la concezione più moderna, derivata dal ripensamento, in termini nuovi, dell'intero problema della gestione dei dati.
Questa nuova concezione, acquisita ormai in sede ufficiale di standardizzazione, è dovuta a W. Bachman, della Honeywell Information Systems, che la propose originariamente agli inizi degli anni '60.
Il principio del database costituisce un salto concettuale nei metodi di gestione dei dati; esso tuttavia trae le sue origini dalla critica dei metodi tradizionali e ne rappresenta il logico superamento.
E' ponendosi in questa prospettiva storica che meglio si può intendere il significato del database. In questo articolo, pertanto, verrà dapprima illustrato come, dalle esigenze che stanno alla base della gestione dei dati, siano state formulate le organizzazioni tradizionali. Successivamente, verrà mostrato come, dall'analisi critica di queste ultime, siano emersi i principi fondamentali del database.

Fig. 1 - Flussi di dati.



2. Le organizzazioni tradizionali

Come è noto, un impiego fondamentale degli elaboratori elettronici si basa sulla loro capacità di immagazzinare, aggiornare e rendere quindi disponibili dati di qualsiasi genere in quantità sempre maggiori con tempi sempre più brevi. E' chiaro quindi l'interesse a cercare, considerando da un lato la problematica dei vari utilizzatori e dall'altro le possibilità messe a disposizione dalle più moderne tecnologie, il modo ottimale con cui strutturare ed immagazzinare l'insieme dei dati nelle varie Unità Periferiche (memorie ausiliarie a nastri o dischi magnetici, lettori di schede, stampanti e così via). Tale insieme di dati, nel dialogo tra l'elaboratore e l'utilizzatore, o meglio tra le memorie secondarie ed i programmi applicativi, può essere visto e quindi associato ad un « flusso » di informazioni di scambio. Per « flusso di dati » si intende, pertanto, un insieme organizzato di informazioni lette da, o registrate su memorie secondarie (fig. 1).
E' nato, quindi, il problema di quali e quante organizzazioni di flusso fosse necessario realizzare al fine di coprire il maggior numero possibile di esigenze dei vari utenti. Senza ripercorrere il processo analitico attraverso cui si è passati, possiamo dire che si è giunti alla definizione di tre organizzazioni di flusso, non tali certamente da coprire l'intera casistica, ma tali che i vari problemi siano ad esse riportabili.
Vediamo quali sono state le principali esigenze che si è cercato di soddisfare in tale ricerca.
Possiamo dire che la problematica relativa ai siste-

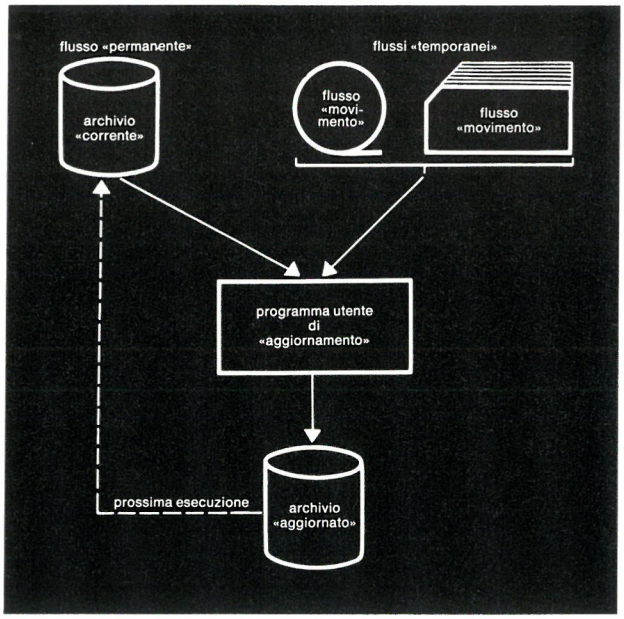


Fig. 2 - Flussi « permanenti » e « temporanei ».

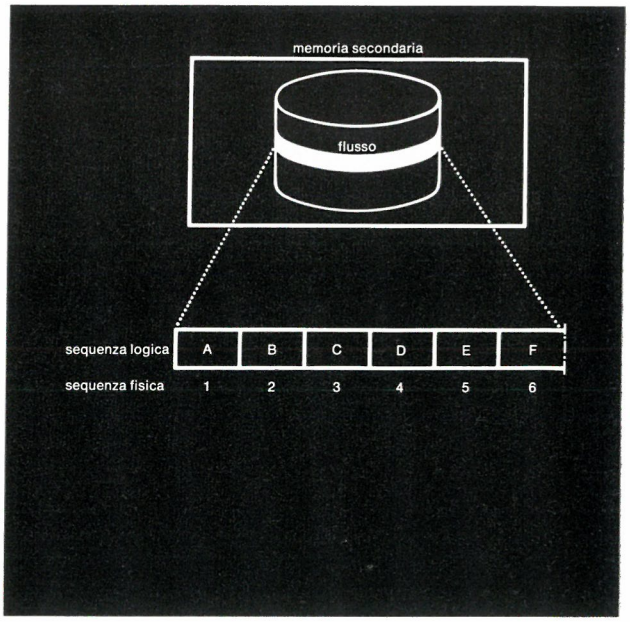


Fig. 3 - Organizzazione sequenziale.

mi di gestione di una azienda può essere schematizzata considerando il seguente ordine di problemi:

- problemi relativi all'organizzazione del sistema di gestione;
- problemi relativi agli strumenti di trattamento delle informazioni.

Dando per scontato che lo « strumento » utilizzato sia l'elaboratore elettronico vediamo il primo punto. L'organizzazione di un sistema di gestione presuppone che siano completamente definiti:

- i risultati cercati
- i dati disponibili
- i trattamenti da effettuare

E' soprattutto relativamente all'ultimo punto, cioè ai *trattamenti da effettuare* che lo studio delle organizzazioni di flusso è stato orientato. Da questi studi è risultato che si hanno in genere due tipi di flusso:

— quelli di tipo « permanente » destinati a contenere dati di base essenziali, come per esempio nome ed indirizzi dei clienti di una banca, numero del conto corrente e così via;

— quelli di tipo « temporaneo » destinati a contenere dati che servono per effettuare aggiornamenti periodici dei flussi precedenti (i cosiddetti « flussi movimento ») ; per esempio il flusso « depositi-prelievi » per l'aggiornamento dei conti correnti dei clienti di una banca, che una volta utilizzati per effettuare i relativi aggiornamenti non hanno più ragione di esistere (fig. 2).

In genere i « flussi movimento » non hanno mai una particolare sequenza secondo cui essere elaborati, si è perciò adottata, per essi, una organizzazione di tipo *« sequenziale »*, dove cioè la sequenza logica secondo cui i vari records componenti il flusso vengono trattati coincide con la sequenza fisica secondo cui sono registrati sui vari supporti (fig. 3). Per i flussi di tipo « permanente » le esigenze, invece, sono di tipo diverso: occorre infatti reperire i records in modo *« diretto »* senza, quindi, una sequenza logica prestabilita. Si è pensato di utilizzare a tale scopo il contenuto di un opportuno campo del record (chiamato campo « chiave ») per permettere l'individuazione della sua posizione all'interno del flusso e quindi il relativo reperimento in modo diretto.

Si hanno, in questo caso, organizzazioni di flusso « dirette » o « random » ed « indiciate ».

Le prime utilizzano gli « indirizzi » dei records forniti tramite il campo « chiave » o direttamente dal contenuto stesso del campo o indirettamente dopo averli ottenuti da un opportuno algoritmo di randomizzazione (fig. 4). Le seconde, sempre basandosi sul contenuto del campo « chiave », accedono ad una parte specifica del flusso, chiamato « indice », che altro non è che una tabella di corrispondenza tra valore della chiave e ubicazione del record stesso nel flusso (fig. 5).

Esistono, soprattutto per queste due ultime organizzazioni, diverse versioni sulla struttura che i dati devono avere all'interno del flusso allo scopo di ottimizzare ora l'una, ora l'altra applicazione, specialmente in funzione della riduzione dei temi di elaborazione; ma la discussione di questo punto non è significativa ai fini di questo articolo.

E' chiaro comunque, che niente vieta di avere organizzazioni con possibilità di accesso « diretto » per i flussi temporanei ed organizzazioni « sequenziali » per i flussi permanenti.

Riassumendo possiamo dire che esistono le seguenti necessità:

- reperire i records sequenzialmente secondo una certa sequenza logica;
- reperire i records in modo diretto formulando:

— l'indirizzo del record stesso (direttamente o indirettamente);

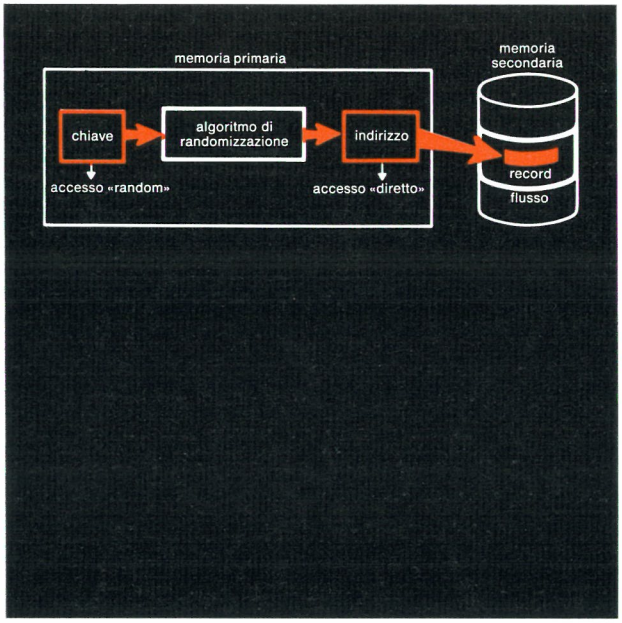


Fig. 4 - Organizzazione « diretta » o «random ».

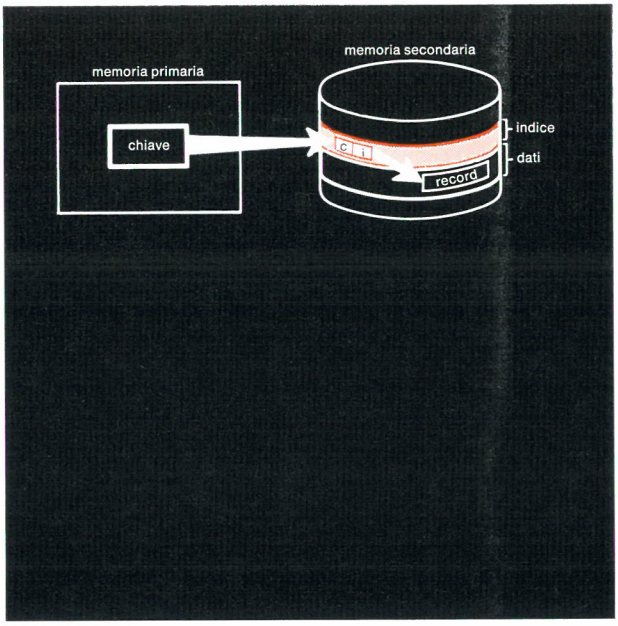


Fig. 5 - Organizzazione indiciata (c = chiave, i = indirizzo).

— un valore univoco che tramite una o più tabelle di corrispondenza permetta di risalire all'indirizzo del record e quindi al record stesso.

3. Problemi connessi alle organizzazioni di tipo tradizionale

Prendiamo ora in esame, per fare un esempio concreto, un sistema di informazione relativo alle vendite e utilizziamo tale esempio per mostrare quali sono gli inconvenienti più grossi e comuni in un approccio di tipo tradizionale. Senza scendere nei dettagli delle possibili soluzioni del problema, si potrebbe pensare di avere sostanzialmente, per coprire le varie esigenze, tre flussi e cioè:

1. Un flusso composto da records « Venditori » con i relativi ordini di acquisto per elaborazioni di tipo generali.

2. Un flusso composto da records « Ordini di Acquisto » per le spedizioni.

3. Un flusso contenente records « Voci di magazzino » per indagini di produzione.

La fig. 6 illustra un approccio di tipo convenzionale mettendo in evidenza il contenuto che potrebbero avere i vari records nei 3 flussi.
E' da notare come la serie di records mostrata nella fig. 6 contenga un considerevole numero di *« dati ridondanti »* (campi in bianco) che, studiando organizzazioni più complesse possono probabilmente essere ridotti, ma mai completamente eliminati.
La presenza di tali informazioni ridondanti comporta inevitabilmente, in caso di aggiornamento delle stesse *l'esecuzione di più programmi o più operazioni di I/O* (Input/Output) per mantenere l'intero archivio (composto in questo caso da tre flussi) allo stesso livello di aggiornamento. Non solo ma l'occupazione stessa dei flussi ne risulta influenzata con conseguente *riduzione dei tempi di accesso causa l'aumento di dimensione degli archivi.*
Inoltre i records, anche all'interno di uno stesso flusso, portano dati in numero variabile; si hanno quindi *records di lunghezza variabile* con relativo appesantimento nell'elaborazione sia da parte del sistema che dei programmi applicativi.
Altra considerazione è che l'utilizzo dei tre flussi prima descritti è sostanzialmente di tipo diverso. Infatti, mentre solitamente il flusso « Ordini d'acquisto » è soggetto ad elaborazioni di tipo sequenziale, il flusso « Venditori » più soventemente viene elaborato in modo « random », cioè partendo da un « Ordine d'acquisto » si risale al corrispondente « Venditore » nell'apposito flusso, la cui sequenza di elaborazione non è perciò aprioristicamente determinabile. Si ha quindi un'organizzazione « sequenziale » per il flusso « Ordini d'acquisto », mentre per il flusso « Venditori » ci si orienta, più propriamente, ad organizzazioni di tipo « diretta » o « indiciata ».
Questo comporta che, avendo le diverse organizzazioni di flusso una chiara influenza sulla logica di programmazione, si hanno programmi applicativi diversi per ciascuna organizzazione con conseguente relativa *differenziazione dell'approccio analitico da programma a programma.* Non solo ma, avendo a volte l'organizzazione influenza anche sul formato stesso dei dati, *aggiornare gli stessi dati nei diversi programmi può comportare modifiche differenti.*
Questi ultimi due punti sono diretta conseguenza dell'avere più di una organizzazione di flusso. Ma, in contrasto con questo, un numero limitato di possibilità di scelte comporta *il non riuscire sempre a trovare la soluzione ottimale per una specifica applicazione o per una particolare elaborazione.*
Considerevoli risultano, inoltre, gli sforzi di analisi e gli studi che devono essere effettuati per usufruire

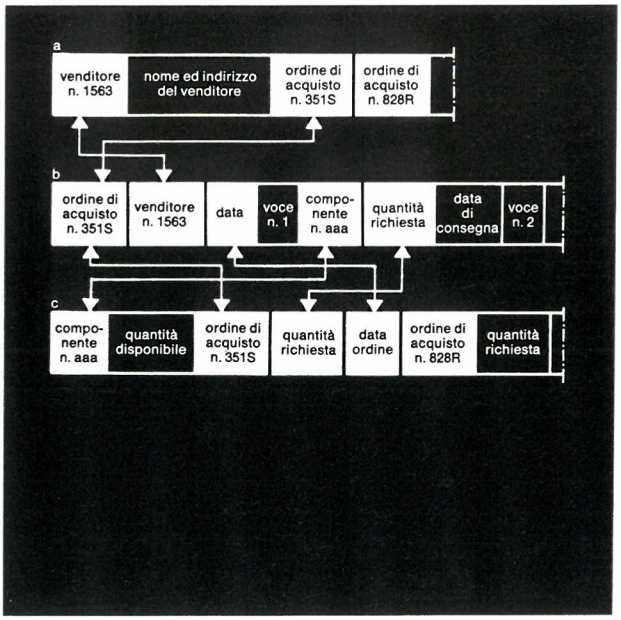


Fig. 6 - Formato di records convenzionali (a = record « venditori », b = record « ordini di acquisto », c = record « voci di magazzino »).

di tutti i vantaggi offerti dal « hardware », nel processo di ottimizzazione dei tempi di I/O. Si introduce così un ulteriore fattore vincolante, cioè: *programmi studiati appositamente per adattarsi a particolari caratteristiche hardware delle unità periferiche,* con inevitabile conseguente perdita di tali vantaggi quando si passa da un tipo di periferica ad un altro (per es. da un tipo di disco ad un altro tipo). Nel caso in cui poi, durante l'evoluzione di una azienda, pur non cambiando la struttura logica degli archivi, si voglia, per esempio, aggiungere un campo nuovo in un flusso o se ne voglia incrementare la lunghezza, si debbono aggiornare tutti i programmi applicativi preesistenti anche se con tali nuove modifiche non hanno niente a che vedere, nel senso che non ne sono né direttamente né indirettamente interessati. In altre parole, si ha una *strettissima dipendenza dei programmi applicativi dal formato dei flussi e dei records che lo compongono,* in quanto la descrizione completa del flusso è sempre parte integrante del programma.
Infatti, qualunque sia il linguaggio di programmazione utilizzato (sia esso COBOL, ASSEMBLER, PL-1 o altri) la descrizione dei flussi elaborati costituisce una parte inscindibile del programma sorgente. Ciò porta, tra l'altro, ad un'ulteriore conclusione e cioè che il programmatore deve avere una visione completa dell'intero archivio; viene, cioè, a mancare la possibilità di ottenere *una visione differenziata dei dati immagazzinati nei vari flussi.*
Va ancora notato che nella concezione corrente esiste un'esplicita connessione tra il flusso ed uno specifico lavoro o applicazione. In altri termini, un insieme di dati non è mai considerato disgiunto dal suo potenziale uso; pur tuttavia, considerando la parte di programma dedicata alla descrizione dei flussi, si fa riferimento soltanto alla descrizione dei campi che compongono i records, cioè alla struttura interna dei records. Della correlazione tra i vari records componenti il flusso non viene fatta menzione in nessuna parte del programma sorgente, anche se nella mente dell'analista tali concatenamenti logici sono ben chiari e già definiti.
Per esempio, nel momento in cui si decide di inserire il campo « Venditore » nel record « Ordine di acquisto » si è implicitamente creata una relazione logica tra questo record ed il record « Venditore ». Si è detto « implicitamente » perché tramite i linguaggi attuali non è possibile esplicitare tale relazione logica. Non resta quindi all'analista-programmatore che servirsi della parte procedurale del linguaggio utilizzato per *programmare e rendere esecutive quelle relazioni logiche che sono comunque presenti nel flusso, ma non sono in altro modo esprimibili.*
Riassumendo, le organizzazioni tradizionali dei dati presentano, ad un esame critico, i seguenti inconvenienti:

- presenza nell'archivio di dati ridondanti con conseguente aumento delle dimensioni del flusso e quindi riduzione delle prestazioni;
- esecuzioni multiple per l'aggiornamento dell'archivio con relativa perdita di tempo;
- necessità, in alcuni casi, di avere records a lunghezza variabile con complicazioni sia nel programma utente che nel sistema;
- problemi di analisi differenti in funzione delle diverse organizzazioni di flusso;
- scelta limitata delle organizzazioni di flusso messe a disposizione in relazione alla vastissima casistica applicativa;
- dipendenza dei programmi applicativi al formato fisico dei flussi e dei records che lo compongono, con necessità di aggiornamenti multipli e differenti di tutti i programmi preesistenti anche per la minima modifica alla struttura dei dati, con relativo aumento dei tempi di codifica, di prova e manutenzione;
- mancanza di una visione « differenziata » dei dati immagazzinati nell'archivio;
- impossibilità di esplicitare le relazioni esistenti tra i vari records affinché sia il sistema e non l'utilizzatore a tenerne conto;
- dipendenza dei programmi da particolari prestazioni hardware con il duplice inconveniente di complicare l'analisi e di perdere, in generale, in efficienza cambiando il tipo di periferica (pur restando nell'ambito della stessa classe).

4. I concetti del Database

Il « database » trova il suo punto di partenza nelle critiche, ora esposte, alle organizzazioni tradizionali degli archivi di dati. I nuovi concetti nascono come superamento degli inconvenienti messi in evidenza. Seguendo questo filo logico, conviene, anzitutto, distinguere, tra gli inconvenienti segnalati, quelli che

sono di tipo « condizionante » da quelli « derivati », i primi essendo, ovviamente, quelli su cui puntare l'attenzione.
Il primo grosso problema di tipo « condizionante » consiste nella presenza di *dati ridondanti* nei diversi flussi. Infatti, a causa di ciò, si è costretti ad avere:

- esecuzioni multiple per gli aggiornamenti;
- pesanti problemi di manutenzione dei programmi applicativi;
- riduzione delle prestazioni;
- records a lunghezza variabile.

Un secondo problema è *la implicita relazione esistente tra un flusso ed il suo potenziale uso senza avere alcuna possibilità di formalizzarla.*
Come terzo punto, si può considerare il fatto che *esiste un numero limitato di organizzazioni di flusso,* con due contrastanti inconvenienti, cioè:

- l'avere più di un'organizzazione di flusso comporta complicazione e diversificazioni nella fase di analisi;
- nel vasto campo delle applicazioni solo alcune di queste possono incontrare una soluzione ottimale, causa la limitata scelta.

Quarto elemento chiave è *la presenza della descrizione completa del flusso come parte integrante del programma stesso,* quindi:

- strettissima dipendenza dei programmi applicativi dal formato fisico dei flussi e dei records;
- mancanza della possibilità di dare una visione parziale e differenziata degli archivi ai vari programmi applicativi.

Quinta ed ultima considerazione è il *dover tener conto delle caratteristiche hardware* per ottimizzare le prestazioni, con relativa:

- complicazione dell'analisi di programmazione;
- stretta dipendenza dei programmi a determinati tipi di periferiche.

Avendo così ristretto il campo di indagine ai problemi essenziali, vediamo ora, come da ciascuno di questi problemi origini un concetto fondamentale del database.

A) Ridondanza dei dati

Cerchiamo di comprendere le cause reali che determinano la lamentata ridondanza dei dati. Allo scopo, occorre chiarire anzitutto alcune questioni di fondo, e cioè: che cosa si intende per dato? sono i dati o qualcosa d'altro ciò che, in ultima analisi, ci serve? e, in questo secondo caso, che funzione svolgono i dati?
In effetti, ciò cui tendiamo è l'*« informazione »* ed i dati sono gli elementi reali e tangibili che la costituiscono, ne sono in un certo senso la concretizzazione.
Essendo, infatti, l'informazione elemento astratto e, quindi, non scambiabile nel dialogo tra memorie primarie e secondarie, si è ricorsi ai *« componenti fisici »* di questo concetto, cioè alle entità reali su

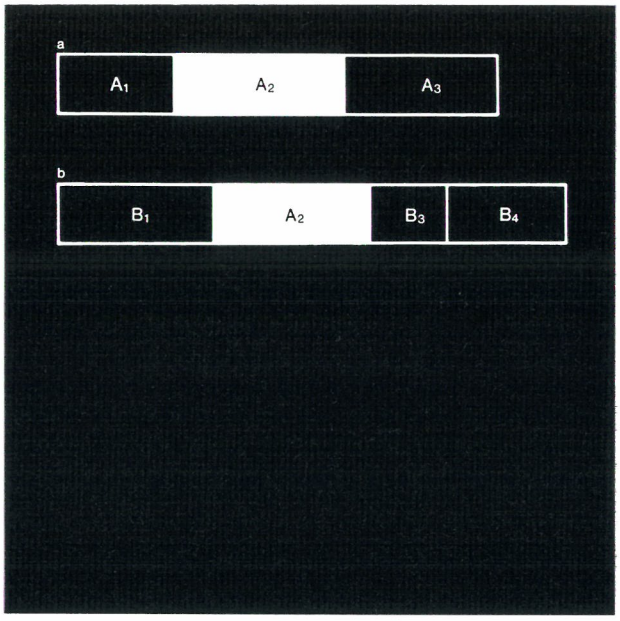


Fig. 7 - Insieme « semplice » di informazioni con ridondanza del campo « A_2 » (a = record di dati facenti parte dell'insieme « A » di informazioni; b = record di dati facenti parte dell'insieme « B » di informazioni).

cui esso si basa, appunto i dati. In altre parole l'informazione ha luogo quando i dati sono interpretati secondo determinate convenzioni di rappresentazione; nasce laddove i dati sono letti e manipolati. Non risiede, quindi, nelle varie memorie ausiliarie se non in forma potenziale.
Infatti, il processo di memorizzazione delle informazioni può essere visto come la fase di scomposizione dell'informazione stessa nei suoi *« dati elementari »* che vengono poi registrati sulle varie periferiche.
Nella struttura delle organizzazioni di tipo tradizionale l'informazione viene fatta coincidere o con il record stesso, nella combinazione dei campi di dati che lo compongono, o con la combinazione dei vari records. Una combinazione cioè di tipo puramente fisico (sequenza fisica all'interno dei records), preordinata (definita all'atto della descrizione del record o del flusso), rigida (non più cambiabile, se non ristrutturando il flusso). L'avere alla base della informazione elementi così strettamente fusi tra di loro porta inevitabilmente alla ripetizione ogniqualvolta un dato entra come componente in più tipi, o « insiemi » di informazioni, oppure è utilizzato come mezzo per colleagre fisicamente tra di loro due o più records che hanno una correlazione logica nella costituzione di un « insieme di informazioni ». Che cos'è effettivamente un *« insieme di informazioni »?* Le informazioni appartengono ad uno stesso « insieme » quando sono generate dagli stessi campi dati elaborati nel medesimo modo. Cioè possiamo, per esempio, parlare di insieme di informazioni « A » quando queste informazioni risultano sempre generate dai campi A1, A2 ed A3. E' possibile, anzi in pratica capita molto di sovente, che uno (o più) campi

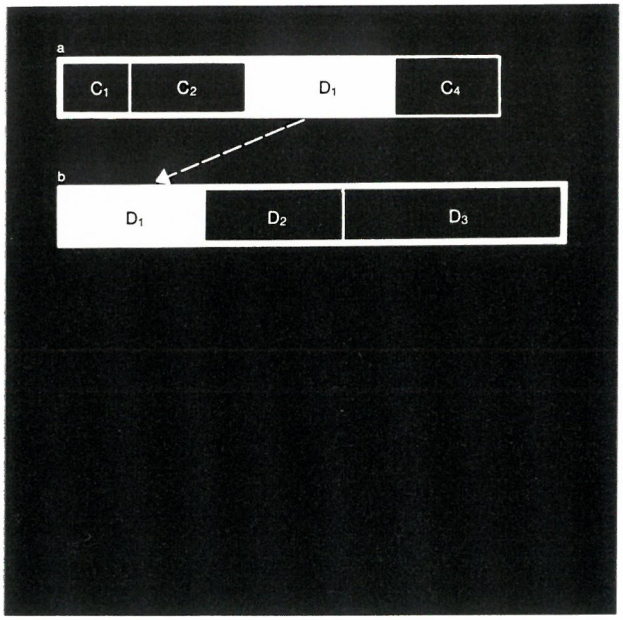


Fig. 8 - Insieme « composto » di informazioni con ridondanza del campo « D_1 » (a e b = records di dati facenti parte dell'insieme « C » di informazioni).

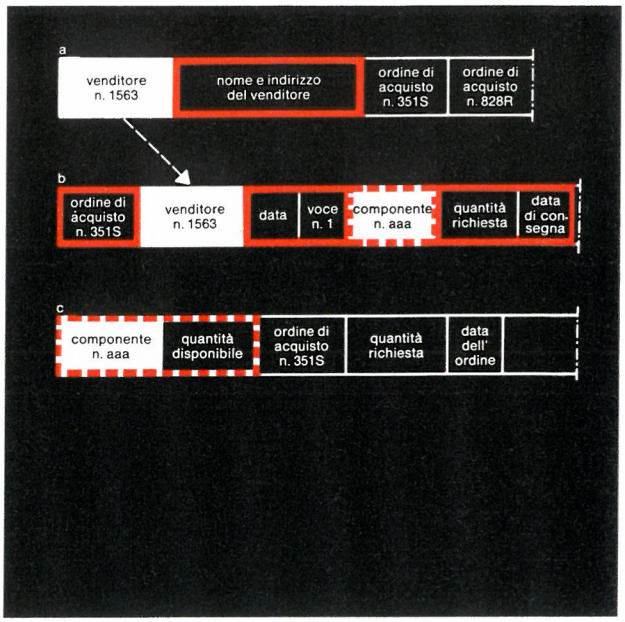


Fig. 9 - Insieme di tipo « semplice » e « composto » con rispettive ridondanze (a = record « venditori », b = record « ordini di acquisto », c = record « voci di magazzino »).

entrino come componenti in un altro insieme di informazioni. Per esempio, l'insieme « B » potrebbe essere composto da B1, A2, B3 e B4. Se, dunque, ora noi raccogliamo i campi A1, A2 ed A3, come facenti parte di record alla base dell'insieme « A » di informazioni, ed i campi B1, A2, B3 e B4, come componenti di un altro tipo di record costituente l'insieme « B » di informazioni, abbiamo creato una ridondanza (il dato A2) (fig 7). Inoltre per determinate elaborazioni, l'insieme di informazioni necessarie è composto dall'intersezione di due o più differenti insiemi di informazioni. Si parla in questo caso, di un « insieme » di tipo « composto » in contrapposizione con i primi che chiameremo « insiemi » di tipo semplice . E' chiaro che nel processo di costruzione delle informazioni « composte » si deve disporre delle informazioni « semplici » e, quindi, dei records che ne sono alla base. Tali records devono essere tra di loro correlati affinché una volta acceduto ad uno di essi, si possa risalire agli altri logicamente associati. Questa condizione porta in genere alla ripetizione nei records di determinati campi, che si possono chiamare campi chiave, tramite cui è possibile individuare la posizione del record voluto all'interno di un flusso.

Per esempio, un insieme di informazioni « C » può avere, come dati componenti, i campi C1, C2, D2 e D3 che però sono raccolti in due tipi di records con conseguente duplicazione di uno di essi, cioè di un campo chiave, che permette di reperire l'altro record ad esso associato (il dato D1) (fig. 8).

Rifacciamoci, per chiarire meglio il concetto sopra descritto, all'esempio preso in considerazione al paragrafo 2.

Tra le possibili elaborazioni che si possono fare prendiamo in considerazione le due seguenti:

- lista totale o parziale degli « Ordini di acquisto » per perparare le tabelle di spedizione;
- lista totale o parziale delle « voci di magazzino » per controllo della situazione magazzino, dopo l'evasione degli « ordini di acquisto ».

Primo punto da considerare è quale « insieme » di informazioni è necessario per ciascuna elaborazione e da qui stabilire quanti e quali sono i dati richiesti per la sua costituzione.

Nel primo caso, i dati di cui abbiamo bisogno sono (fig. 9):

— numero dell'ordine di acquisto;
— nome ed indirizzo del venditore;
— data dell'ordine;
— voce numero 1 con il componente richiesto e rispettiva quantità;
— voce numero 2 con il componente richiesto e rispettiva quantità;
—
—
— voce numero « n » con il componente richiesto e rispettiva quantità;
— data di consegna

(Questi campi sono contornati da linea rossa continua nella fig. 9).

Per la seconda elaborazione i dati necessari potrebbero essere:

— nome del componente;
— quantità disponibile

(Questi campi sono contornati da linea rossa tratteggiata nella fig. 9).

Rifacendosi alle definizioni prima date, si può dire che nel primo caso è richiesto un insieme di informazioni di tipo « composto ». Infatti, per avere i dati necessari alla elaborazione richiesta devono essere

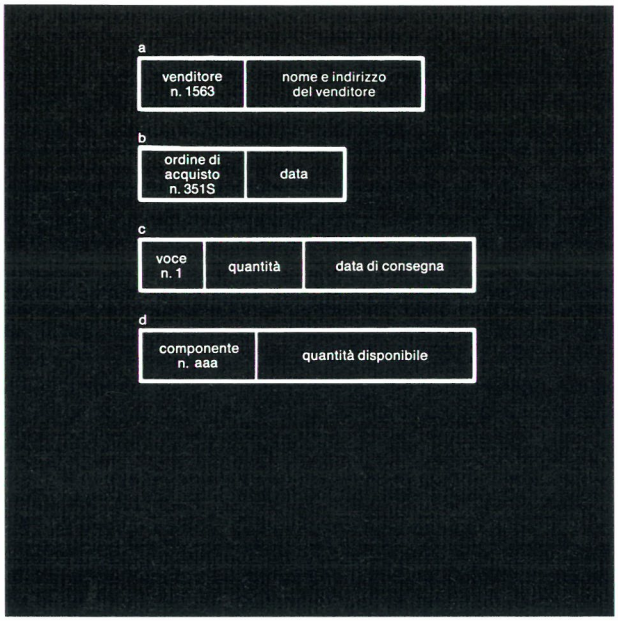


Fig. 10 - Formato di record database *(a = record « venditore », b = record « ordine di acquisto », c = record « voci per ordine di acquisto », d = record « voci di magazzino »).*

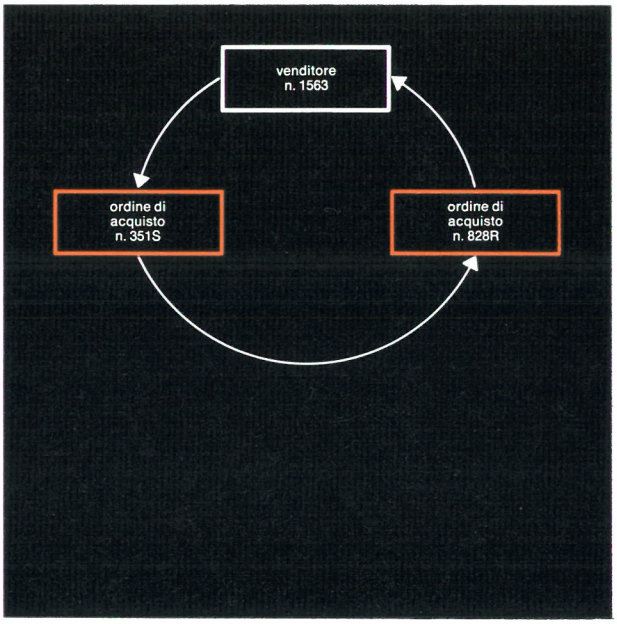


Fig. 11 - Una « catena » con un « master » *e due* « details ».

letti due records: un record « ordini d'acquisto » e, tramite il campo « numero di venditore », un record del flusso « venditori » per risalire al nome ed indirizzo. Questo comporta la *ripetizione del campo chiave* « numero di venditore » nei due flussi considerati (il « numero di venditore » è necessario nel record « venditori » per altri tipi di elaborazione). Nel secondo caso si ha, invece, un insieme di informazioni di tipo « semplice » in quanto i dati necessari sono tutti reperibili all'interno dello stesso record « voci di magazzino ». Abbiamo, però, un dato (« nome del componente ») che partecipa a tutti e due gli insiemi di informazioni con conseguente *duplicazione del dato* stesso nei due rispettivi records. Abbiamo, in precedenza, fatto distinzione tra informazione e dato. Vediamo come si traducono in pratica questi concetti.

Cerchiamo, cioè, di riconoscere nei nostri flussi tradizionali i dati che sono necessari e sufficienti a tutti gli insiemi di informazioni richieste; creiamo cioè i *dati di base* (da cui appunto il nome « database ») per tutte le possibili elaborazioni.

Come primo passo cessiamo di pensare in termini di records, di formati di flussi, di organizzazioni, e portiamoci a vedere i problemi in termini di insiemi di informazioni richiesti e, quindi, in termini di dati elementari, visto che sono questi gli elementi basilari della informazione. Così facendo i dati di base dell'esempio risultano quelli mostrati in fig. 10. Abbiamo, cioè, eliminato i campi ridondanti, sia relativi ad insiemi di informazioni « semplici » che « composte », e nella ricerca dei dati fondamentali per la futura informazione, abbiamo spezzato in due il record « ordine di acquisto » creandone uno nuovo chiamato « voci per ordine di acquisto ». Infatti, essendo tali « voci » in numero variabile rendevano di lunghezza variabile anche il vecchio record « ordine di acquisto ».

Tra l'altro, quindi, in questo processo di risalita ai dati elementari, abbiamo anche ottenuto di avere records di lunghezza fissa all'interno di ciascun tipo. Vediamo ora in che modo, da questo insieme slegato di dati, si passa ad una struttura organizzata.

B) Relazione tra flusso ed il suo uso potenziale

La seconda componente dell'informazione, come abbiamo visto, risiede nella combinazione strutturata, nella relazione esistente tra i vari records. Abbiamo constatato che, nelle organizzazioni tradizionali, nel momento stesso in cui l'analista definisce un certo formato dei dati e dei records, ne ha già stabilito le interrelazioni logiche, cioè il modo in cui i vari records devono combinarsi tra loro per ottenere i risultati voluti. Ciò nonostante, nella « descrizione del flusso » c'è solo il modo di definire la struttura all'interno dei records e le varie categorie dei campi dei dati, per cui tutto quello che riguarda il trattamento legato alle varie combinazioni dei records, deve essere esplicitato nell'apposita parte procedurale del programma.

Visto quindi, che nella descrizione di un flusso è insito il suo uso potenziale, vediamo in che modo tali *« relazioni logiche »* si possono esplicitare per far sì che sia il sistema e non il programmatore o l'analista a tenerne conto. La soluzione di tale problema è riposta nel significato intrinseco di quei « campi chiave », di cui si è parlato nel paragrafo precedente, relativamente agli insiemi « composti » di informazioni.

Constatato che questi campi, gestiti totalmente dal-

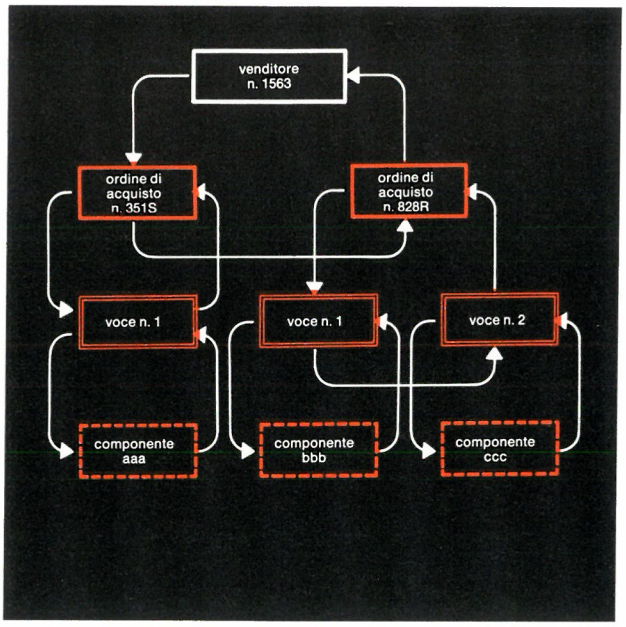


Fig. 12 - Esempio di database.

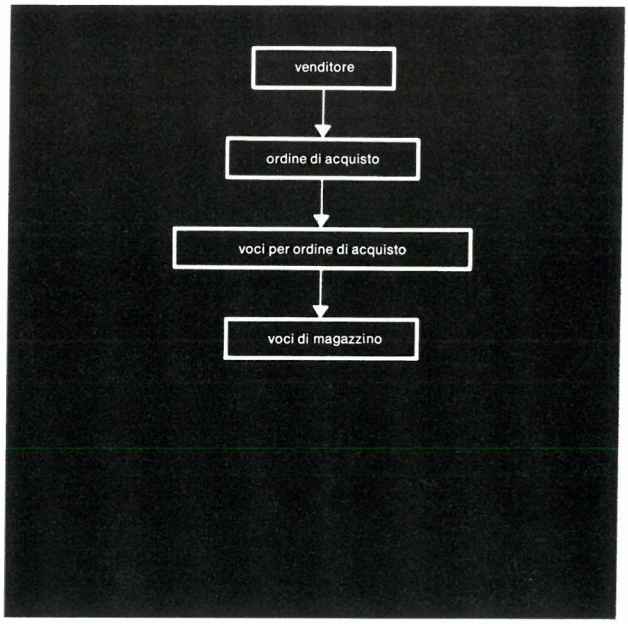


Fig. 13 - « Struttura dati » dell'esempio fatto.

l'utilizzatore, servono per realizzare un concatenamento fisico tra due records a valle di una riconosciuta relazione logica, perché non permettiamo all'utilizzatore di « esplicitare » nella descrizione dei suoi records tale relazione? Sarà poi il sistema ad occuparsi di creare e gestire un « qualcosa » che realizza il concatenamento voluto, sollevando, di conseguenza, l'analista-programmatore da tutti i problemi procedurali che abbiamo prima visto. Anche il momento in cui effettuare tale dichiarazione è il più opportuno, in quanto coincide con la descrizione dei records componenti il flusso che, abbiamo fatto più volte notare, porta implicitamente tali correlazioni.

E' nato così il concetto di « catena », intesa come un insieme di records tra di loro associati e portanti dati logicamente dipendenti nella costituzione di un insieme di informazioni.

Così, rifacendoci sempre all'esempio di prima, il record « venditore » entrerà con il record « ordine di acquisto » a costituire una « catena » in quanto entrambi contengono dati che si integrano tra di loro per costituire un certo insieme di informazioni. Esiste, inoltre, tra i due records un rapporto non solo di tipo qualitativo, ma anche quantitativo, nel senso seguente. Per ogni « venditore » non esiste, in genere, un solo « ordine di acquisto », ma ne esisteranno un certo numero tutti associati allo stesso « venditore ». Questo tipo di relazione numerica all'interno della catena porta a classificare diversamente i componenti della catena stessa. Si chiama *« master »* (alcuni preferiscono il termine *« owner »)* il record principale di una catena, quello che condiziona la presenza di tutti gli altri (nel nostro esempio il record « venditore »), mentre gli altri records, che possono essere in numero variabile (i records « ordini di acquisto »), si chiamano *« details »* (alcuni preferiscono il termine *members »)* (fig. 11). Possiamo adesso completare la definizione di catena che, quindi, può essere intesa come un insieme di records tra di loro associati, consistenti in uno ed un solo record « master » ed in zero, uno o più records « details ».

Continuiamo, ora, nel nostro esempio dal punto dove ci eravamo fermati. Eravamo arrivati all'individuazione dei dati e, quindi, dei records elementari costituenti il database; applichiamo adesso il concetto di catena. Per fare ciò, possiamo seguire questo ragionamento già prima iniziato: per ogni « venditor » ci possono essere più « ordini di acquisto », ogni « ordine di acquisto » può contenere più voci, ciascuna delle quali fa riferimento ad una « voce di magazzino ». Ed ecco create le varie catene che compongono il database dell'applicazione presa in considerazione (fig. 12).

Più in generale possiamo dire che, l'insieme delle catene che compongono un flusso di tipo database ne costituiscono quella che viene chiamata « struttura dei dati ».

C) Diagrammi della « struttura dei dati »

Esiste un modo sintetico per rappresentare le « strutture dei dati ». I due elementi base da rappresentare sono i « records » e le « catene ». I primi si rappresentano con un *rettangolo,* i secondi con una *freccia.* Vale a dire che in un diagramma di questo tipo un rettangolo rappresenta una « classe di records » mentre una freccia rappresenta una « classe di catene ». Seguendo questa semplicissima norma la « struttura dei dati » dell'esempio considerato è quella mostrata in figura 13.

Occorre tener presente che il rettangolo posto all'origine della freccia rappresenta una classe di records, ciascuno dei quali è un record « master » di una catena che appartiene alle classi di catene rappresentate dalla freccia. Il rettangolo alla fine della freccia rappresenta una classe di records, di cui zero,

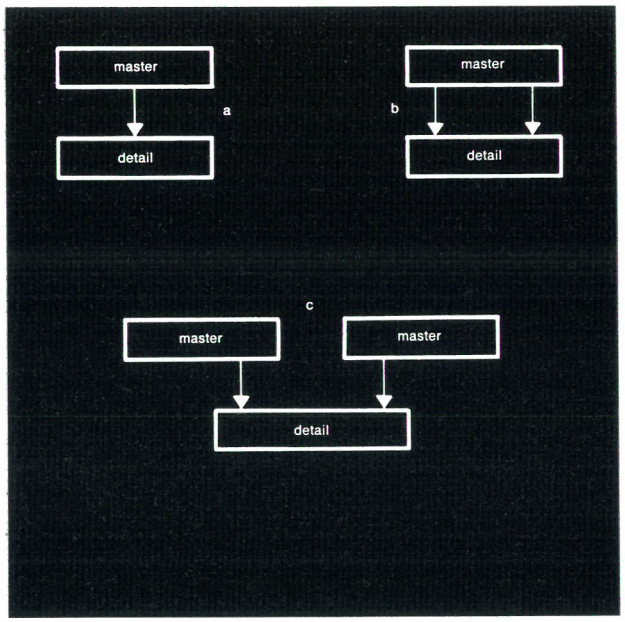


Fig. 14 - Tipi di strutture (a = struttura gerarchica, b = struttura a reticolo semplice, c = struttura a reticolo composto).

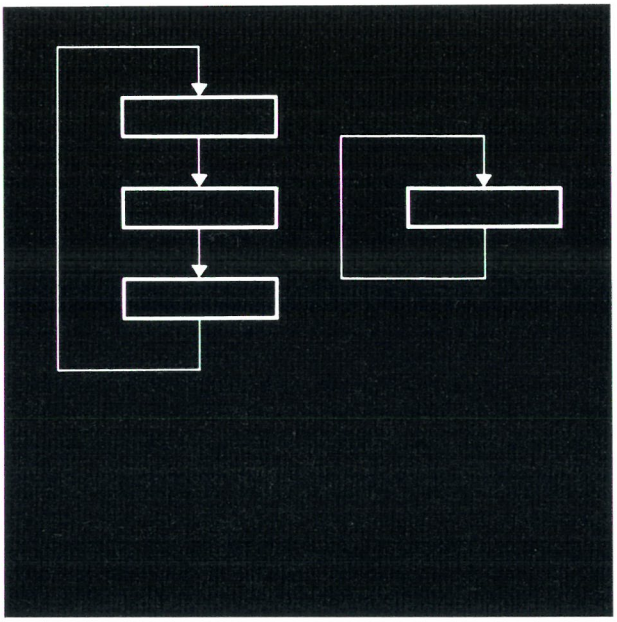


Fig. 15 - Strutture non ammesse.

uno o più sono records « details » di una catena che appartiene alla classe di catene rappresentate dalla freccia.

E' da notare che questi diagrammi non contengono né implicano informazioni sul numero delle catene, né sul numero dei records « details » che ciascuna catena può contenere. Volendo avere una rappresentazione della struttura precedente che tenga anche conto di questi fattori, ricadremmo nell'esempio della fig. 11, che, come si può notare, è molto più dispersivo, in special modo quando si hanno strutture complesse. La struttura dei dati che abbiamo visto è solo un esempio di come i vari records possono entrare in combinazione tra di loro. Records che sono « details » in una catena possono essere « masters » in un'altra, cioè, più in generale, un record può entrare come detail « n » volte in « n » catene diverse ed « m » volte come master in altrettante « m » catene.

A seconda poi delle varie combinazioni, si possono avere « strutture gerarchiche » (fig. 14 a), strutture a « reticolo semplice » (fig. 14 b) e strutture a « reticolo composto » (fig. 14 c).

E' chiaro che ciascuna di queste strutture può entrare in qualsiasi combinazione ed un numero qualsiasi di volte a comporre strutture più complesse e generali. Senza entrare in ulteriori dettagli basta dire che, praticamente, non esistono limitazioni sulle possibili combinazioni dei vari records tra di loro (in alcuni testi tali combinazioni sono chiamate anche « strutture extra-records »). La sola considerazione da fare, per altro intuitiva, è che l'unica struttura dati non accettabile sul database si ha quando un tipo di record master diventa detail di se stesso, direttamente o indirettamente (fig. 15).

D) Rigidità delle organizzazioni tradizionali

Abbiamo visto che nei metodi tradizionali esistono tre tipi di organizzazione: sequenziale, diretta o random ed indiciata.

Tali organizzazioni rispondono senz'altro ad effettive necessità applicative, ma oltre a fornire un campo abbastanza limitato di scelta, costringono, a causa della diversità di struttura, ad un processo di analisi di tipo differenziato. Infatti, una volta decise le organizzazioni su cui i dati vanno riportati si devono affrontare problematche molto diverse in funzione dei vari formati di flusso.

Abbiamo anche visto come nel database si sia praticamente ricorsi ad una *integrazione* dei diversi flussi componenti l'archivio in un unico flusso unificando così la fonte da cui prelevare le informazioni. In questa fase di integrazione viene ridata, quindi, omogeneità all'analisi, in quanto l'organizzazione è diventata, ora, unica ed unico è perciò il tipo di struttura e la problematica connessa.

Nell'individuazione dei dati di base di una generica azienda, o, più in generale, di un utilizzatore, e nella susseguente raccolta di tali dati in un unico flusso, viene spontaneo domandarci se hanno perso di significato le organizzazioni sequenziali, dirette, indiciate, random, che pure avevamo riconosciuto essere nate da reali ed effettive necessità.

Naturalmente non hanno perso di significato; vediamo allora come il database ne tiene conto.

Nella presentazione dei fondamenti delle organizzazioni tradizionali avevamo riconosciuto esistere principalmente tre differenti modi di accesso al record:

- sequenziale, tramite la sequenza in cui i record sono stati registrati;
- diretto, tramite l'« indirizzo » del record, cioè fornendo al sistema un valore opportuno che permette

di individuarne la posizione all'interno del flusso;

• indiciato, tramite un campo chiave che permette di risalire alla corrispondenza record e suo indirizzo.

E' chiaro che le stesse possibilità sono richieste anche in questa nuova gestione dei dati; mentre però negli approcci tradizionali questi metodi di accesso danno luogo ad altrettante organizzazioni, nel database la loro realizzazione è ottenuta tramite la creazione di tre « tipi » di records. Sono stati, cioè, creati dei « tipi » di records che avessero implicito il modo di accesso e quindi, anche, il modo di inserimento nel flusso.

Si è così arrivati ai:

• records *« secondari»* , cui si può acecedre solo tramite la loro appartenenza ad una o più catene, cioè durante lo scorrimento *sequenziale* di uno o più records associati alla stessa catena;

• records *« primari »*, cui si può accedere direttamente, fornendone la posizione logica all'interno del flusso (il « reference-code » o la « database-key »);

• records *« calcolati »*, l'accesso ai quali avviene utilizzando, da parte del sistema, il contenuto di un opportuno campo, allo scopo di risalire alla posizione del record del flusso. Questo campo, scelto dall'utilizzatore tra i campi dati dal record, viene spesso chiamato *campo chiave.*

Va inoltre aggiunto che il proiettare i metodi accesso, non su diverse e distinte organizzazioni, ma su differenti tipi di records, che possono far quindi parte dello stesso flusso, è coerente col processo di unificazione dei dati di base, che risulta, anzi, ulteriormente perfezionato.

Infatti, in tale processo, senza nulla togliere alle varie possibilità di accesso ai records, si è semplificata la fase di analisi, che risulta ora omogenea. Allo stesso tempo, ricordando come la struttura dei dati sia composta da catene variamente associate tra di loro in una vastissima gamma di possibilità, si è ampliato il campo in cui ricercare la scelta ottimale per le diverse applicazioni.

Possiamo, a questo punto, definire il database come un *« unico insieme di dati integrati, raccolti in tre diversi tipi di records tra di loro variamente associati in catene ».*

E) Descrizione del flusso incorporato nei programmi applicativi

Nella concezione tradizionale della gestione dei dati i programmi applicativi sono studiati su specifici modelli di organizzazione, diventandone così rigidamente dipendenti, tanto che la descrizione stessa del flusso ne costituisce parte integrante. Non solo ma, se più programmi accedono allo stesso flusso, pur elaborando dati diversi, devono contenere l'intera descrizione dei dati costituenti il flusso, anche di quelli, cioè, cui non sono assolutamente interessati. Per di più tali descrizioni di flusso si moltiplicano a seconda del numero delle applicazioni e delle organizzazioni utilizzate.

Nel database, al contrario, si ha un approccio ai problemi di tipo globale e si può riportare nella struttura la totalità delle necessità applicative. Si viene, cioè, a sostituire alle varie descrizioni dei flussi con le loro diverse organizzazioni, la definizione e la relativa descrizione dell'intero, unico, flusso database che raccoglie tutti i dati elementari necessari alle varie applicazioni.

Quindi, non più « n » diverse descrizioni di flusso, ma una sola.

Questo non vuol dire, naturalmente, che nel database ciascun programma applicativo deve « portarsi dietro » la descrizione totale dei dati; anzi è possibile, non solo che le varie applicazioni « vedano » quella parte di database a cui sono direttamente interessati, ma che tale descrizione parziale del flusso non faccia parte integrante del programma sorgente. In altre parole è possibile avere programmi applicativi contenenti solo una descrizione parziale del flusso, ignorando, al limite, completamente la totale struttura dei dati. Per di più si può far sì che tale descrizione sia fatta separatamente dal programma applicativo e venga soltanto da questo « invocata » al momento in cui è richiesta.

E' facile, a questo punto, intuire come non solo risulti realizzata la possibilità di « visibilità differenziata » del database dai vari utenti a diverso livello, ma anche l'eliminazione dell'interdipendenza fra programma applicativo e memorizzazione fisica dei dati. Vale a dire che l'aggiunta di nuovi campi, di nuovi dati o di nuove catene non si riflette in nessuna modifica ai programmi già esistenti, a meno che non ne siano direttamente interessati.

Per concludere questo punto, riferiamo che, nella terminologia ufficiale il linguaggio tramite cui è possibile definire separatamente la struttura del database viene indicato con DDL (Data Description Language), mentre relativamente al trattamento dei dati, nella parte procedurale del programma, si ha il DML (Data Manipulation Language).

F) Dipendenza dei programmi applicativi dalle caratteristiche hardware

A questo proposito, basta ricordare che in ambito database esiste una parte di software, generalmente associata al sistema operativo, chiamata DBMS (Data Base Management System) che si preoccupa di tener conto delle varie caratteristiche hardware al fine di ottimizzare le prestazioni totali.

G) Riepilogo

Riassumendo, le caratteristiche fondamentali e innovative del database rispetto alle soluzioni tradizionali sono le seguenti:

— fusione ed integrazione dei vari dati tradizionalmente sparsi sui diversi flussi in un unico flusso generale, costituente la base dei dati che sono indi-

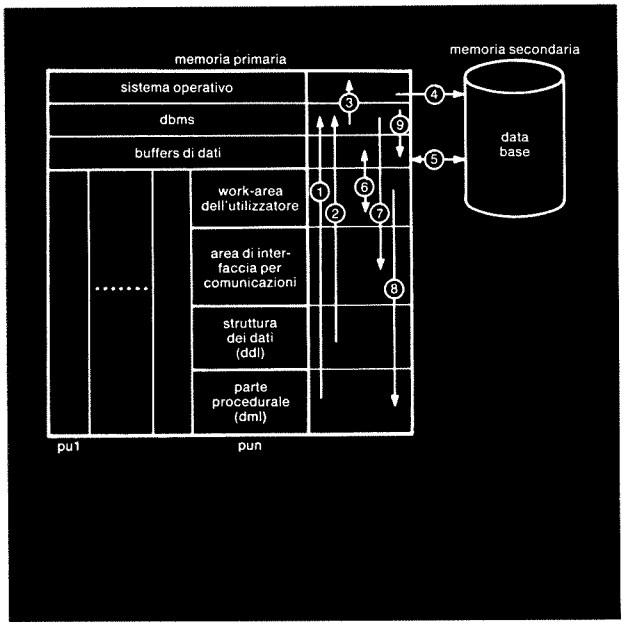


Fig. 16 - Data base management system (dbms) (pu = programma utilizzatore 1,2 ... n).

spensabili per la costituzione delle « informazioni »;

— formalizzazione delle relazioni esistenti tra i dati tramite il concetto di « catena », di « master » e di « detail », che offrono praticamente un campo vastissimo di possibili combinazioni tra di loro;

— descrizione della « struttura dei dati », cioè delle varie catene e loro combinazioni tramite un linguaggio apposito e non facente parte integrante dei programmi sorgente (DDL);

— inclusione nel DBMS dei problemi derivanti dalle caratteristiche hardware delle periferiche.

Il funzionamento di un sistema di ambito database può essere agevolmente seguito utilizzando lo schema di fig. 16, in cui la sequenza logica di elaborazione è indicata dai numeri crescenti e dalle frecce associate.

1. Il programma utente comanda la ricerca (o l'inserimento) di un dato tramite il DML;

2. il DBMS analizza la richiesta e produce informazioni complementari che riguardano quella parte di struttura di database coinvolta nella richiesta. Queste informazioni sono ricavate dalla struttura dei dati, che ora in fase di esecuzione, è diventata una parte del programma oggetto;

3. ottenute queste informazioni il DBMS interfaccia il sistema operativo per effettuare quelle operazioni di I/O necessarie per espletare la richiesta fatta;

4. il sistema operativo effettua le operazioni fisiche sul flusso database;

5. il sistema operativo trasferisce dati utilizzando uno o più buffers (memorie di transito);

6. il DBMS trasferisce i dati tra i buffers e particolari work-area del programma;

7. il DBMS trasferisce, anche, informazioni relative all'operazione effettuata; tali informazioni sono disponibili all'utente tramite opportune aree di comunicazione del programma stesso;

8. i dati sono ora a disposizione per ogni tipo di elaborazione;

9. il DBMS ha il controllo completo dei buffers, che sono comuni a tutti i programmi che accedono al database.

5. Conclusione

Il concetto di database, come risulta da quanto esposto, rappresenta un fondamentale passo avanti nella organizzazione degli archivi di dati ed il superamento di tutte le concezioni precedenti.

Ma il database non è soltanto una brillante teoria; esso costituisce ormai una realtà concreta adottata da costruttori ed utilizzatori di elaboratori elettronici. La Honeywell, da cui è originato il concetto, è stata anche pioniera nella applicazione pratica del database col sistema IDS (« Integrated Data Store »), sul mercato da oltre 10 anni.

Alla luce dei fatti, non è improprio considerare il database come una pietra miliare dell'informatica ed una svolta nella utilizzazione dei sistemi di elaborazione elettronica.

Bibliografia

[1] Feature Analysis of Generalized Data Base Management Systems, *CODASYL Systems Committee*, 1971.

[2] Data Base Task Group Report to the CODASYL Programming Language Committee, 1971.

[3] Charles W. Bachman, « Data Structure Diagrams », *DATA BASE, A Quarterly Newsletter of SIGBDP*, 1, n. 2, 1969.

[4] Charles W. Bachman, « The Evolution of Data Structures », *Procs. of the NordDATA 73 Conf.*, Copenhagen, 1973.

[5] IFIP Guide to Concepts and Terms in Data Processing, North-Holland Publishing Company, Amsterdam, 1971.

[6] Integrated Data Store, Order n. BR69, *Honeywell Information Systems*, 1972.

La protezione giuridica del software

CARLO FALCETTI
Honeywell Information Systems Italia
Servizio Brevetti

1. Introduzione

E' una caratteristica comune di tutti i paesi moderni di possedere una normativa che si propone di incentivare lo sviluppo scientifico, tecnologico, culturale e artistico della società, mediante il riconoscimento e la tutela di determinati diritti che vengono riconosciuti a coloro che contribuiscono a tale sviluppo.
Secondo il particolare tipo di contributo portato alla società, si attribuiscono diritti diversi. Agli autori di opere letterarie, musicali, figurative, si riconoscono diritti d'autore; agli inventori di apparecchiature o procedimenti industriali, si riconoscono diritti di brevetto.
L'articolo che segue affronta il problema se il software come opera creativa intellettuale goda di tutela giuridica e di quale tutela.

Per quanto sia ormai più di un decennio che si discute, soprattutto all'estero, se i programmi per gli elaboratori elettronici godano di qualche tutela giuridica, la risposta all'interrogativo rimane ancora un ambiguo « ni ».
Fondamentalmente, la ragione di questa incertezza è dovuta a due fattori: da un lato, il fatto che i diversi istituti giuridici esistenti nelle legislazioni nazionali per la tutela di certe categorie di prodotti dell'ingegno hanno ovviamente una portata giurisdizionale limitata territorialmente; dall'altro, il fatto che il software è un prodotto intellettuale in cui si associano e hanno rilievo sia aspetti formali che sostanziali e di struttura.
Secondo che si intenda proteggere l'uno o l'altro aspetto, o ambedue, si può ritenere conveniente il ricorso all'uno o all'altro dei diversi istituti giuridici esistenti.
Non tutte le legislazioni nazionali concordano però nel riconoscere la tutelabilità del software mediante i diversi istituti. Anzi, in molti paesi, per diversi istituti ciò è esplicitamente escluso, quando non è dibattuto e controverso.
Ne deriva che una risposta all'interrogativo proposto presuppone la definizione dell'area territoriale e dei termini o dello scopo della tutela a cui ci si vuole riferire.
Se poi si considera che il « software » è facilmente riproducibile o duplicabile con dei costi trascurabili ed è facilmente trasferibile, si conclude che una protezione adeguata può aversi solo nella misura in cui una effettiva tutela giuridica può estendersi su diversi paesi e ciò rende ancora più incerto un giudizio sulla efficacia della protezione quand'anche si arrivi alla conclusione che effettivamente il software gode di una tutela giuridica, almeno in certi paesi.

2. Caratteristiche del Software

Come è noto, il « software » (in contrapposizione all'« hardware » che indica gli elementi materiali di un sistema di elaborazione di dati) può essere definito come l'insieme di informazioni astratte che concorrono alla gestione di un calcolatore elettronico e al suo impiego per la risoluzione di singoli problemi oppure come un insieme di procedure logiche e informazioni formalizzate in modo da poter essere comprese ed eseguite da un calcolatore elettronico.
In relazione alle funzioni specifiche cui il software è destinato si usa fare la distinzione tra software applicativo e software di base.
Il software applicativo è costituito da programmi di lavoro o programmi utenti, cioè dagli effettivi programmi che l'utente di un sistema di elaborazione dati ha interesse che siano eseguiti, quali il controllo di un processo, l'elaborazione degli stipendi del personale, l'aggiornamento dei registri delle scorte di magazzino, le fatturazioni, ecc.
Il software base è costituito da tutto quell'insieme di programmi, compilatori, assemblatori, supervisori, che corredano uno specifico sistema di elaborazione dati e gli consentono di svolgere i programmi utenti.
In particolare, i programmi supervisori provvedono ad organizzare il sistema di elaborazione, gli danno una fisionomia, lo gestiscono, intervono in caso di errori, operano insomma come intelligenza astrat-

ta di supervisione perché il sistema esegua in modo corretto il compito o i compiti che gli sono stati affidati.
Mentre i programmi applicativi possono essere scritti ignorando completamente, o quasi, la reale struttura in cui dovranno essere svolti e con linguaggi di programmazione di tipo universale, quali FOR TRAN, COBOL, ecc., il software base, e in particolare i programmi supervisori, sono intimamente connessi a questa, e ogni tipo di sistema di elaborazione richiederà pertanto il suo proprio programma supervisore e, più in generale, ogni tipo di calcolatore avrà il suo software di base.
Con l'avvento di sistemi di elaborazione sempre più potenti e sofisticati e di nuove tecniche di impiego (multiprogrammazione - time sharing), il software va assumendo un ruolo sempre più preponderante. Si stima che in questi ultimi anni il costo di progetto e definizione del software di base per un calcolatore sia almeno eguale, se non superiore, al costo di progettazione e sviluppo dell'hardware del calcolatore stesso. Anche per quanto riguarda il software applicativo si può ritenere, senza timore di commettere un errore grossolano, che le spese annuali di sviluppo e acquisizione di tale prodotto sostenute annualmente dagli utenti, siano come ordine di grandezza eguali alle spese sostenute per l'acquisto o il noleggio dell'hardware.
In ogni caso, una caratteristica comune del software base e del software applicativo è l'elevato costo di definizione, ossia di analisi dei problemi da risolvere e dei compiti da svolgere, di redazione del programma, di correzione e verifica di operabilità, contrapposto alla estrema facilità ed economicità con cui il prodotto, una volta definito può essere riprodotto, copiato, utilizzato indefinitamente.
In un certo senso, il costo del software è tutto un costo di progettazione del prodotto, mentre il costo di riproduzione è del tutto trascurabile.
Ciò giustifica l'interesse dimostrato congiuntamente, anche se temperato da perplessità, sia dai fabbricanti di hardware, in quanto produttori di software di base, sia dagli utenti e dalle case produttrici di software, prevalentemente applicativo, per l'identificazione di possibili forme di protezione che impediscano la contraffazione e l'impiego abusivo di questo tipo di prodotto.

3. La brevettabilità del Software

La più estesa forma di protezione accordata dalle diverse legislazioni nazionali alle opere dell'ingegno, ossia alle opere creative, è costituita dal brevetto per invenzione.
Il brevetto per invenzione è un diritto di monopolio limitato nel tempo, che la legge riconosce ad alcune categorie di opere creative, precisamente alle invenzioni industriali.
Diritto di monopolio, anzitutto nel senso di « ius escludendi alios » per cui a nessuno è permesso, senza autorizzazione del detentore del brevetto, di mettere comunque in atto l'invenzione protetta, anche se si perviene posteriormente al trovato inventivo in modo autonomo e creativo, cioè senza copiare e averne preventiva conoscenza.
In secondo luogo, per metterne in evidenza la forza e la portata, diritto di sostanza e non di forma, nel senso che tutto quanto ricade nell'ampia idea di soluzione oggetto del brevetto, indipendentemente dagli aspetti formali che la rappresentano o che la esemplificano, rientra in questo diritto di monopolio.
Non tutto è brevettabile, ma solo quei trovati che soddisfano a determinati requisiti che per la legge italiana, e per la maggior parte delle legislazioni brevettuali estere, consistono nella novità estrinseca nell'apporto inventivo o originalità o novità intrinseca, nella utilità, e nella industrialità.
Ora è fuori di discussione che il software sia utile, possa essere nuovo estrinsecamente (un programma mai scritto prima) possa essere originale (nuovo intrinsecamente o non ovvio).
Se il software goda del requisito di industrialità, questo invece è controverso e tanto più in quanto il concetto di industrialità non è un concetto definito in modo inequivocabile.
A chiarire questo concetto provvede entro certi limiti e con esemplificazioni l'art. 2585 del nostro Codice Civile: « Possono costituire oggetto di brevetto le nuove invenzioni atte ad avere una applicazione industriale quali un metodo o un processo di lavorazione industriale, una macchina, uno strumento, un utensile o un dispositivo meccanico, un prodotto o un risultato industriale e l'applicazione tecnica di un principio scientifico, purché essa dia immediati risultati industriali. In quest'ultimo caso il brevetto è limitato ai soli risultati indicati dall'inventore ».
In altre parole o l'invenzione gode del requisito della materialità o è rivolta indirettamente ad agire sul mondo materiale in modo da ottenere oggetti di produzione.
La dottrina e la giurisprudenza italiane sono unanimi nell'interpretare la legge nel senso di considerare privi del requisito di industrialità i sistemi di calcolo, i metodi di organizzazione del lavoro, i metodi di misura, i metodi di gestione aziendale.
In questa luce se consideriamo un programma come un modo particolare di elaborazione delle informazioni, o di esecuzione di calcoli, si può concludere immediatamente che il software non è brevettabile.
Di fatto la materia trattata, la materia prima e il prodotto risultante, sono e rimangono delle informazioni. Se non che, si dice, il software è anche quel complesso di istruzioni logiche che dirige il comportamento di un calcolatore elettronico, al punto che senza programmi questo non è in grado di operare. Da questo punto di vista il software agisce direttamente su un prodotto industriale e gli dà una sua fisionomia.
L'hardware cioè, è una macchina che potenzialmente è in grado di trasformarsi in tante macchine di-

verse che operino in modo prefissato, con funzioni di calcolo, di regolazione, di registrazione ecc. secondo il programma che le specializza e le caratterizza per tali compiti.
E dunque i programmi sono qualcosa suscettibile di un risultamento industriale diretto e come tali dovrebbero rientrare nella categoria delle invenzioni brevettabili quando gli altri requisiti (novità intrinseca ed estrinseca) siano verificati.
E' questa la tesi sostenuta per esempio da Ettore Luzzatto [1], e un po' più cautamente da Fabrizio De Benedetti [2].
Tuttavia, in opposizione a questa tesi si può rilevare che un calcolatore elettronico non è un insieme di parti non correlate tra loro, che acquistano fisionomia e unità solo quando un programma dà loro una identità, bensì un insieme di parti correlate tra loro e correlabili in certi modi, statici e dinamici definiti dall'insieme di istruzioni e sequenze di istruzioni che possono essere comprese ed eseguite da un calcolatore.
Per fare un esempio banale, possiamo comparare l'hardware a un regolatore a camma, per esempio un regolatore di temperatura, del tipo di quelli impiegati nella strumentazione industriale che consente di variare e controllare la temperatura di un forno nel tempo, per sottoporre dei materiali a opportuni trattamenti termici.
La temperatura è imposta dal profilo della camma modificando il quale, si modificano le caratteristiche di regolazione.
Nei limiti imposti dalle caratteristiche dello strumento (per es. massimo e minimo diametro ammissibile della camma, massima pendenza) si può scegliere tra un numero praticamente infinito di profili di camma, tagliando questi secondo le esigenze specifiche. Un profilo può quindi essere nuovo ma non per questo trasforma il regolatore in qualcosa di nuovo e inventivo, né appare verosimile che possa essere considerato brevettabile perché consente di realizzare una variazione di temperatura secondo una legge prefissata: se l'effetto di tale variazione è un nuovo trattamento termico che conferisce ai materiali proprietà nuove, non è brevettabile il profilo della camma ma il trattamento termico, indipendentemente dal profilo della camma che eventualmente lo rappresenta.
Così si può concludere anche per un programma di elaboratore elettronico, che può essere comparato al profilo della camma di regolazione e come questa non deve essere ritenuta brevettabile.
Se per caso il programma è diretto alla regolazione di un processo industriale, e da tale regolazione ne deriva un nuovo processo industriale o un miglioramento di un processo industriale noto, non per questo il programma assurge al livello di industrialità: industriale e brevettabile rimane il processo nuovo e migliorato che viene realizzato sotto le direttive del programma.
La distinzione fatta da alcuni (E. Luzzatto) tra software per regolazione di processi industriali e software di tipo commerciale, per sostenere la brevettabilità indiscutibile di almeno il primo, mi pare dunque irrilevante ai fini della brevettabilità.
Semmai alla luce di alcune decisioni della nostra magistratura si potrebbe sostenere la brevettabilità dei programmi, in quanto eventualmente tali da portare al migliore funzionamento, con maggiore efficienza, del calcolatore.
Per esempio la Corte di Cassazione ha sostenuto (sentenza n. 2632 del 2 ottobre 1949) che « non v'è dubbio che il metodo o processo brevettato possa condurre ad un risultato che non consista nella produzione di un bene materiale o di un dato prodotto... la protezione del brevetto per invenzione può essere chiesta non soltanto quando il trovato si materializza in un determinato bene o in un determinato prodotto, ma anche se si configura come di ordine immateriale, se di interesse tipicamente industriale vale a dire se è mezzo di applicazione industriale.
E possono quindi fondamentalmente esemplificarsi come suscettibili di tutela le invenzioni che tendono ad un risparmio dell'energia e del calore, ad un acceleramento di una reazione chimica, all'elevamento del grado di tensione elettrica, *al migliore funzionamento di una macchina,* all'abbreviazione dei tempi di lavorazione ».
Un calcolatore elettronico è una macchina, un prodotto industriale e il suo migliore impiego ha certamente un rilievo industriale, poco importando che il calcolatore sia impegnato nell'esecuzione di operazioni contabili o nella regolazione di un processo produttivo.
Senonché, accettando senza limitazioni tale conclusione, che il software goda in toto del requisito dell'industrialità si arriva al paradosso che la giurisprudenza riconosce come brevettabile ciò che per un altro verso esclude dalla brevettabilità.
La giurisprudenza, ha infatti stabilito, senza ombra di dubbio, che sono privi del requisito dell'industrialità i sistemi di calcolo ad uso scientifico, e i sistemi di controllo e regolazione della produzione e dello svolgimento dell'attività tecnico-commerciale dell'impresa quali per esempio i sistemi di lavorazione in serie e i metodi di registrazione contabile.
A dispetto del fatto che un programma può essere formalizzato in modo da realizzare una migliore utilizzazione dell'hardware, non bisogna dimenticare che scopo delle procedure contenute nei programmi è, con le eccezioni che vedremo, l'elaborazione di informazioni astratte in cui rientra, guarda caso come applicazioni più tipiche, l'esecuzione di calcoli scientifici, la registrazione contabile, il controllo e la regolazione della produzione o dello svolgimento dell'attività tecnico-commerciale dell'impresa.
In sostanza i programmi consentono di attuare mediante macchine delle procedure logiche che prima erano compiute mentalmente e registrate manualmente.
Si arriva quindi al paradosso che un nuovo metodo

di registrazione contabile, un nuovo metodo di calcolo, una nuova procedura logica, di per sé non brevettabili, diventano brevettabili, in quanto eventualmente consentono una migliore utilizzazione di una macchina.
La contraddizione può forse essere risolta se si ammette che la portata della protezione brevettuale accordata a un programma sia limitata al suo impiego in elaboratori elettronici, per esempio imponendo una opportuna formulazione delle rivendicazioni ossia delle clausole riassuntive che definiscono in un brevetto l'estensione o portata dell'invenzione e i limiti entro i quali viene accordata la protezione brevettuale.
In tal modo sarebbe possibile salvaguardare i fondamentali principi di libertà di pensiero e di comportamento umano che stanno alla base della esclusione di brevettabilità per le attività puramente intellettuali e comportamentali.
In proposito è interessante osservare come si è sviluppato questo orientamento per esempio negli Stati Uniti dove il problema della brevettabilità del software è stato ed è tuttora dibattuto.
Già nel 1966 un rapporto della Commissione presidenziale incaricata di studiare l'eventuale riforma del sistema brevettuale americano raccomandava l'esclusione dei programmi dalla protezione brevettuale. La proposta era ripresa nel « Patent Reform Bill » S. 1042 presentato al Congresso nel 1967 dal senatore Mc. Lennan. Il § 106 di tale testo legislativo era specificatamente dedicato alla esclusione dei programmi dalla protezione brevettuale.
La proposta fece naturalmente rumore, e suscitò aspre reazioni negative, e vuoi per calmare le acque e dare soddisfazione ai fautori della brevettabilità, vuoi perché si riteneva che la brevettabilità dei programmi dovesse esseer definita sulla base di criteri più generali, le proposte di riforma legislativa presentate successivamente non hanno più toccato l'argomento.
Da parte sua il « Patent Office » pubblicava in via preliminare sulla « Official Gazette » del 6 luglio 1966 delle « Guidelines » per l'esame di domande di brevetto relative a programmi.
A seguito dei commenti ricevuti, riformulava e rendeva esecutive tali Guidelines con pubblicazione nella Official Gazette del 22 Ottobre 1968. In sostanza la posizione assunta dal Patent Office era la seguente:
« I processi mentali non sono brevettabili. Un processo brevettuale nel senso statutorio della legislazione brevettuale (USA) deve essere qualcosa che agisce su elementi materiali e produce cambiamenti nel loro carattere o nella loro condizione. Pertanto un programma che produce solo un risultato numerico, statistico o informativo non è brevettabile. Tuttavia se il programma è combinato in modo non ovvio con altri elementi in modo da produrre un risultato fisico, per esempio per dirigere il taglio di una lamiera o simili, può essere brevettabile in quanto parte di un processo ». Conclusione: i programmi in sé non si ritengono brevettabili, anche se descritti in termini di apparato. Eventuali programmi di processo sono invece brevettabili in quanto processi e non in sé (come dire che anch'essi come programmi non sono brevettabili).
Questi criteri erano appunto stati seguiti dal Patent Office per respingere nello stesso periodo domande di brevetto per programmi.
Tra queste si citano come significative per i seguiti giudiziari i casi « Prater & Wei » e « Bernard Fetter » (1).
La domanda presentata da Prater & Wei riguardava un metodo e apparato per la determinazione dei componenti di una miscela gassosa mediante l'elaborazione opportuna, con un programma di calcolatore, dei dati spettografici relativi a tale miscela.
Le rivendicazioni coprivano sia il metodo, descritto in termini di generazione di segnali rappresentativi di dati e di elaborazione di tali segnali, ossia di calcolo, sia l'apparato descritto in termini di mezzi generici per l'esecuzione delle operazioni previste dal metodo.
Nell'ambito del Patent Office, l'Esaminatore prima e poi il Board of Appeal, respingevano la domanda di brevetto adducendo che il metodo rivendicato poteva essere svolto mentalmente o manualmente con carta e matita avendo per oggetto dati di partenza e dati risultanti e quindi non era brevettabile.
L'elaborazione dei dati mediante calcolatore è cosa ovvia, pertanto anche le rivendicazioni di apparato, espresse in termini così generali da coprire un calcolatore programmato in modo da effettuare l'elaborazione prevista dal metodo, erano respinte.
Insoddisfatti della decisione del Patent Office i proponenti ricorrevano alla Magistratura e la « Court of Customs and Patent Appeals » rovesciava in parte la decisione del Patent Office affermando che effettivamente le rivendicazioni di metodo erano così ampie da coprire l'esecuzione mentale del metodo e quindi erano da respingere (ma in principio, un'altra formulazione, tale da escludere l'esecuzione mentale del metodo, le avrebbe rese accettabili) mentre le rivendicazioni di apparato, riguardando un calcolatore programmato in un modo intrinsecamente nuovo e originale, dovevano ritenersi brevettabili.
Oltre tutto la domanda di brevetto conteneva una descrizione dettagliata di almeno una apparecchiatura utilizzabile per eseguire il metodo: non si poteva quindi sostenere che la domanda fosse inaccettabile per difetto di descrizione (decisione resa il 14 agosto 1969).
La domanda di brevetto presentata da Bernard Fetter riguardava un metodo e apparato per la genera-

(1) Prater & Wei: Domanda n. 155,236 presentata il 20 novembre 1967 per « Riduzione di dati risultanti da analisi spettrali ». Bernard Fetter: Domanda n. 151,909 presentata il 13 novembre 1961 per « Metodo e apparato per la realizzazione automatica di una rappresentazione piana di un oggetto tridimensionale ».

zione di una immagine piana di un oggetto tridimensionale. Mediante equazioni è possibile stabilire una relazione tra le coordinate tridimensionali di ciascun punto di un oggetto e le coordinate bidimensionali della proiezione di ciascun punto dell'oggetto su un piano qualsiasi. Utilizzando un calcolatore e un « plotter » o tracciatore di grafici è possibile elaborare tali equazioni e comandare il plotter in modo da ottenere la rappresentazione piana dell'oggetto secondo un qualsiasi punto visuale.

La novità di tale « invenzione » consisteva essenzialmente nella formulazione delle particolari equazioni da usare essendo indifferente la scelta del calcolatore e del plotter e in proposito nessuna descrizione di hardware era fornita nella domanda.

Le rivendicazioni erano espresse in termini di apparato comprendente un calcolatore programmato per risolvere tali equazioni e un plotter per convertire in disegno i segnali generati dal calcolatore, nonché in termini di procedimento consistente nel programmare un calcolatore in modo da risolvere quelle particolari equazioni.

Anche in questo caso l'Esaminatore prima e il « Board of Appeal » poi respingevano la domanda di brevetto adducendo che la combinazione strutturale calcolatore-plotter era già nota e che la novità risiedeva esclusivamente nelle equazioni ossia in qualcosa che può essere elaborato mentalmente e quindi non è brevettabile.

Il proponente la domanda di brevetto adduceva che le rivendicazioni presentate definivano il *nuovo uso* di una combinazione strutturale nota, e quindi l'invenzione era brevettabile, ma anche su questo punto il Patent Office replicava che nuovo era semmai il risultato e non l'uso, essendo cosa nota che i calcolatori possono essere usati per risolvere equazioni.

Insoddisfatto della decisione del Patent Office il proponente ricorreva alla magistratura e la « Court of Customs and Patent Appeals » rovesciava la decisione del Patent Office sostenendo che per quanto l'aspetto inventivo sostanziale risiedesse nelle equazioni, queste non dovevano essere considerate disgiuntamente dal contesto di impiego indicato dalle rivendicazioni. La limitazione introdotta nelle rivendicazioni con il riferimento all'impiego di un calcolatore salvaguardava il principio del libero godimento e impiego di tali equazioni in procedimento mentali e manuali, e proponendo una alternativa meccanica di risoluzione forniva il presupposto di industrialità necessario alla concessione di un brevetto. Inoltre un calcolatore programmato in un modo nuovo e non ovvio, per quanto non dovesse essere considerato come una nuova macchina costituiva pur sempre un nuovo e utile miglioramento di una macchina non programmata. Pertanto la concessione doveva semmai essere condizionata dalla verifica di sussistenza dei requisiti di novità estrinseca ed intrinseca. E a questo proposito, la Corte, pur ammettendo che la maggior parte dei programmi possa essere considerata ovvia, non ha escluso che alcuni programmi possano avere un che di inventivo o di originale. (Decisione resa il 20 novembre 1969).

Si delineava così un chiaro atteggiamento della magistratura statunitense, che riconosceva nel software il requisito della industrialità e condizionava quindi la brevettabilità alla sussistenza degli altri requisiti, quali la novità intrinseca e la novità estrinseca, nonché ad una appropriata formulazione delle rivendicazioni.

Questa è la conclusione di compromesso a cui la magistratura è arrivata oltre che negli Stati Uniti, anche in altri paesi come la Gran Bretagna e la Germania e che ha visto in generale schierati, nei ricorsi di vario grado che hanno avuto luogo, da un lato gli uffici centrali brevetti, sostenitori della non brevettabilità dei programmi, e dall'altra gli inventori e le compagnie produttrici di software.

Si tratta però a ben guardare di un compromesso più che altro apparente e tale comunque da condizionare in modo determinante l'impiego delle procedure brevettate, se si tiene conto del fatto che gli orientamenti attuali nel campo scientifico, amministrativo, gestionale, sono di meccanizzare e automatizzare quanto più possibile le operazioni, per cui una protezione brevettuale, limitata all'impiego di procedure e programmi in calcolatori, di fatto esclude una libera utilizzazione pratica di tali procedure.

Di questa contraddizione si è resa conto la stessa magistratura statunitense, che decidendo in merito alla domanda di brevetto di Benson e Tabbot nel novembre 1972, e a livello di Corte Suprema, riconosceva che l'invenzione non era brevettabile in quanto si riduceva ad una formula matematica per la conversione di numeri decimali espressi in codice binario in numeri binari puri.

Tale formula, trovava pratica applicazione solo nei calcolatori, e non aveva senso la sua applicazione mentale.

Pertanto accordare protezione brevettuale a un programma che utilizzasse tale formula significava di fatto escluderne ogni possibilità di libero impiego e brevettare la formula in se.

Affrontando più in generale il problema della brevettibilità dei programmi la Corte Suprema affermava esplicitamente che tale decisione non doveva interpretarsi come sanzione della non brevettibilità degli stessi, in merito alla quale una azione legislativa (pro o contro) di chiarificazione era necessaria e lasciava quindi aperto il problema, rimandandone la soluzione al potere legislativo e riservando alla magistratura, sulla base della legislazione vigente, la responsabilità di decidere caso per caso.

Mentre quindi in alcuni paesi, pur condizionandola ad una opportuna formulazione delle rivendicazioni, la brevettabilità dei programmi o almeno di certi programmi sembra ammessa, in altri paesi invece, la brevettibilità dei programmi è stata esplicitamente esclusa o con provvedimenti legislativi (Francia, riforma della legislazione brevettuale) o con restrizioni amministrative degli uffici centrali brevetti (Canadà, Giappone) che richiedono conferma da parte della

magistratura mediante eventuali ricorsi in appelli, o con sentenze di alcune corti (Austria, Australia, Olanda, Svizzera), peraltro non a livello di corte suprema e quindi suscettibili di revisione.
La ragione di fondo nel riconoscere la non brevettibilità dei programmi è stata anche qui la difficoltà di distinguere tra procedimenti logici, non brevettabili, e procedimenti logici eseguiti con macchine.
Per quanto riguarda l'Italia, mancando praticamente ogni esame di merito inventivo delle domande di brevetto, è possibile depositare domande ed ottenere brevetti per programmi di calcolatori, brevetti la cui validità appare tuttavia incerta, almeno fino a quando la Magistratura non sarà chiamata a pronunciarsi in merito sulla base della legislazione attuale, oppure provvedimenti legislativi di riforma della legislazione vigente non chiariranno questo punto. [3]
Per quanto riguarda il cosiddetto Software di base, e in particolare gli « operating systems » il discorso è leggermente diverso. I sistemi operativi, sono programmi destinati alla gestione dell'hardware e sono intimamente connessi a questo.
Un sistema operativo, è organizzato in funzione delle caratteristiche dell'hardware che deve gestire ed è destinato a questo scopo esclusivo: si preoccupa di garantire che tutte le parti del sistema fisico siano coordinate nell'esecuzione di programmi utenti (applicativi) e siano funzionanti, provvede opportune segnalazioni in caso di guasti, di errori, supervisiona globalmente il comportamento dell'hardware.
Tali programmi complessi a differenza dei programmi applicativi devono tener conto della effettiva struttura fisica del calcolatore, e non possono essere considerati e quindi descritti e compresi in un testo brevettuale disgiuntamente dallo specifico sistema di hardware a cui si riferiscono.
Sistema hardware e sistema operativo costituiscono dal punto di vista funzionale un tutto unico, e la copertura brevettuale del sistema operativo mediante rivendicazioni richiede necessariamente l'enunciazione di caratteristiche strutturali a cui il processo o programma si collega.
Sulla base di quanto dianzi accennato, il requisito della industrialità per i sistemi operativi appare dunque molto più solido, e si potrebbe dire fuori discussione, sia in Italia che all'estero.

4. Portata e limiti della protezione brevettuale

Si è visto che per quanto l'orientamento prevalente delle legislazioni nazionali o della loro interpretazione, sia nel senso di escludere la brevettibilità del software, in alcuni paesi, con adeguate limitazioni delle rivendicazioni, ciò è ammesso.
Non è quindi fuor di luogo considerare brevemente quale può essere la efficacia di tale tipo di protezione e se veramente soddisfa le esigenze di protezione dei produttori di software.
Si è già detto che i diritti di brevetto sono diritti territoriali di esclusiva o di monopolio, limitato nel tempo, che proteggono l'ampia idea di soluzione proposta dall'inventore indipendentemente dai dettagli realizzativi.
Il diritto di monopolio nel caso di una macchina o un oggetto è tale che non si può produrre o usare tale macchina o oggetto senza autorizzazione del titolare del brevetto, (limitatamente al paese in cui il brevetto è in forza), e non si possono nemmeno produrre o usare oggetti e macchine simili, che rientrino nell'ampia idea di soluzione, coperta dal brevetto.
La distinzione tra produzione e impiego di una macchina mette in evidenza che, in caso di contraffazione, il titolare di un brevetto può rivalersi sia sul contraffattore che produce la macchina sia sull'utilizzatore.
Così il titolare di un brevetto italiano per una macchina non potrà perseguire eventuali costruttori all'estero della stessa macchina sulla base del brevetto italiano, ma potrà impedire l'importazione, la vendita e l'utilizzazione della macchina contraffattrice in Italia.
Il diritto di monopolio nel caso di un processo o metodo è tale che non si può utilizzare il processo o metodo, o processi o metodi simili che rientrino nell'ampia idea di soluzione coperta dal brevetto.
Così nel caso dei programmi, quando essi risultino coperti da brevetto che li rivendica in termini di apparato programmato per svolgere un compito prestabilità, o in termini di metodo per ottenere determinati risultati, l'eventuale soggetto perseguibile di contraffazione non è né il fabbricante di hardware, né colui che scrive e vende il programma, bensì il solo utilizzatore.
Inoltre, un brevetto di programma non protegge il documento scritto che descrive il processo o apparato programmato, e nemmeno protegge il programma registrato su un supporto (nastro magnetico, schede perforate o altro) in quanto tale.
Infatti il fatto che un programma sia contenuto su un supporto non significa per questo che il programma sia eseguito.
Ciò significa che un brevetto di programma lascia la più ampia libertà di ricopiare e duplicare e distribuire il programma brevettato, e attribuisce il rischio e la responsabilità della contraffazione all'utilizzatore.
L'efficacia della protezione brevettuale nel caso dei programmi, o almeno della maggior parte dei programmi appare quindi estremamente limitata richiedendo l'identificazione, assai difficile, e la persecuzione dei singoli utilizzatori che attualmente compiono la contraffazione.
Il fatto che in molti paesi valga il principio della contraffazione contributoria, per cui chi distribuisce un programma che descrive un processo brevettato (si presume per farlo usare) è ritenuto passibile di concorso in contraffazione, e quindi perseguibile, non semplifica le cose.
Infatti i mezzi di duplicazione e le vie di distribuzione di programmi sono molteplici e difficilmente controllabili. La protezione brevettuale dei programmi agli effetti pratici è quindi inefficace e costituisce semmai una remora all'impiego dei calcolatori e una

spada di Damocle pendente sulla testa degli utilizzatori.
Certamente essa è malvista dai costruttori di hardware, i quali nel corso di trentanni hanno sviluppato macchine sempre più sofisticate e suscettibili di impieghi sempre più generali, il cui uso potrebbe essere seriamente ipotecato e condizionato da una diffusa e indiscriminata applicazione della protezione brevettuale di programmi.
Fortunatamente, quand'anche si ammetta che i programmi godano del requisito dell'industrialità, non per questo sono tutti brevettabili, dovendosi identificare in ciascuno di questi il requisito della novità e della originalità.
La maggior parte dei programmi non sono affatto originali e derivano in modo scontato con modifiche opportune da programmi già scritti o da schemi logici usuali, per cui si può senz'altro sdrammatizzare il rischio accennato.
Inoltre, nel caso di programmi brevettati, non è affatto avveniristico pensare che, con l'avvento del « Time-Sharing » e la realizzazione di sistemi di informatica interconnessi mediante reti telefoniche e ponti radio in cui più centri di elaborazione sono distribuiti su un'area multinazionale, sia possibile eludere una protezione brevettuale trasferendo le informazioni da elaborare in un centro di calcolo collocato in un paese dove i programmi non sono brevettabili, o dove eventualmente quel particolare programma non è brevettato, e ivi effettuare l'elaborazione utilizzando quel particolare programma.
Queste considerazioni, che derivano, l'una dai requisiti generali di brevettabilità, l'altra dal carattere territoriale della protezione accordata, ripropongono la inefficacia della protezione brevettuale nel caso dei programmi.
D'altra parte l'esigenza di una qualche forma di protezione rimane sentita: un programma per quanto ovvio, per quanto non inventivo, comporta pur sempre un costoso lavoro di analisi, di compilazione, di « debugging ». Non è giusto che una volta scritto, un programma, indipendentemente dalla sua originalità, possa essere copiato con una spesa modesta e utilizzato abusivamente.
Nel seguito di questo articolo vedremo pertanto se le legislazioni nazionali prevedono altre forme di protezione più indicate, e i loro eventuali limiti.

Bibliografia

[1] E. LUZZATTO: *Protezione giuridica del Software*, in Atti del Convegno indetto dalla FAST sui « Brevetti di invenzione » a Milano il 5-7 marzo 1969 pag. 101-107. Editore: Il Saggiatore - 1969.

[2] FABRIZIO DE BENEDETTI: *I programmi dei calcolatori: sono essi proteggibili?*, in « Rivista della Proprietà Industriale e della Concorrenza » nn. 1-2 (gennaio, marzo; aprile, giugno) 1968.

[3] Parere in linea di massima negativo sulla brevettabilità del Software esprime anche Gerardo Santini: « La tutela giuridica della programmazione elettronica » in Giurisprudenza Italiana vol. CXX 1968 parte IV - 225-232.

L'elaborazione elettronica nel settore bancario in Italia

VIRGILIO MARANDELLA
Honeywell Information Systems Italia
Direzione Centrale Marketing

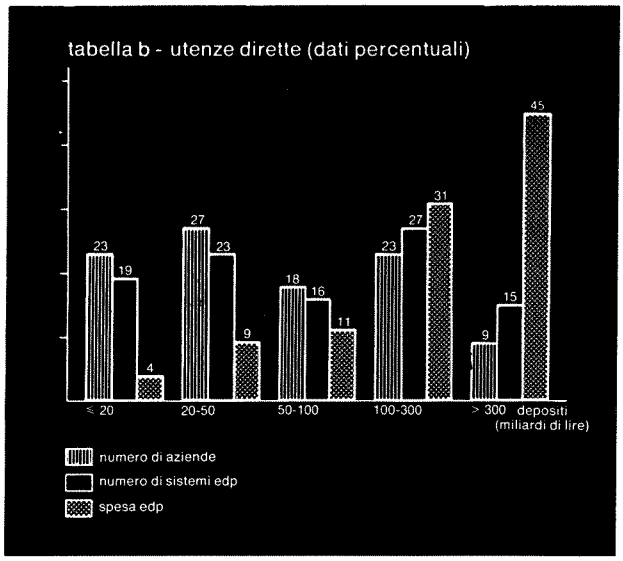
Il livello di automazione raggiunto nel settore delle Casse di Risparmio e delle Banche Popolari Italiane (1) presenta, alla fine del 1973, una notevole consistenza, sia come parco macchine di elaborazione dati, che come forme applicative.
Premesso che il fenomeno della meccanizzazione con elaboratori elettronici non è di manifestazione recentissima (delle 166 utenze dirette attuali censite nella nostra rilevazione, ben 128, e cioè il 77%, si presentavano già meccanizzate alla fine del 1968 con un proprio sistema EDP), il livello attuale di penetrazione dell'elaborazione dati riguarda il 78% delle banche intervistate, mentre soltanto un numero molto esiguo (4 aziende) non presenta a tutt'oggi alcuna forma di utilizzazione di tecniche automatiche di elaborazione dati. Tra questi estremi, si trova poi un discreto numero di aziende (20% del totale) la cui forma di meccanizzazione avviene tramite centri esterni all'azienda stessa, generalmente supportati da altri istituti di credito, da centri servizi specializzati nel settore, o da forme di associazione consortile di interesse prevalentemente locale.
Il numero delle aziende che si avvalgono di queste ultime forme di supporto risulta ancora modesto, mentre senz'altro più diffusa appare la forma di appoggio a centri servizi specializzati.
Il fattore che sicuramente sembra costituire l'elemento discriminante di siffatta distribuzione è senz'altro quello dimensionale, mentre non ci è sembrato determinante, ai fini della spinta alla meccanizzazione con sistemi EDP, quello della localizzazione geografica. In rapporto alla struttura dimensionale, va rilevato che, a meno di 6 aziende, tutte quelle con un ammontare di depositi superiore ai 20 miliardi risultano meccanizzate direttamente, mentre quelle della fascia inferiore, che rappresentano una quota attorno al 50%

(1) L'universo di ricerca è stato definito sulla base dell'Annuario delle Aziende di Credito e Finanziarie edito dalla Associazione Bancaria Italiana; l'oggetto dell'indagine è costituito dalle 213 aziende che, sulla base del Bilancio 1972, presentavano un ammontare di depositi non inferiore ai 5 miliardi di lire.

Tabella A

Area Geografica	Classe di Depositi in Miliardi (%)					
	<20	20-50	50-100	100-300	>300	Totale
Nord	27.8	77.3	100.0	100.0	100.0	72.8
Centro	69.6	100.0	90.9	100.0	100.0	86.2
Sud	63.6	100.0	-	100.0	100.0	80.5
Totale	49.4	90.0	96.7	100.0	100.0	77.9

del totale, manifestano un più recente approccio alla automazione EDP.
E' proprio in questa fascia di aziende che si è avuto il più alto numero di nuove utenze nel periodo 1968-73, soprattutto nelle regioni centrali e meridionali.
Secondo le zone geografiche, nel 1968 si avevano le seguenti quote di aziende meccanizzate direttamente: nel Nord 65%, nel Centro 64% e nel Sud 42%. A 5 anni di distanza, il divario tra Nord e Sud sembra essersi completamente annullato (Tabella A).
La penetrazione EDP, infatti, sia a livello globale che, soprattutto, per le classi dimensionali più piccole, è nettamente più alta nel Centro e nel Sud rispetto a quella delle regioni settentrionali.
La indubbia connessione tra dimensione aziendale ed esigenze di automazione fin qui emersa poteva senz'altro ritenersi scontata, anche a livello intuitivo, in un mercato (come già visto) la cui formazione non risale agli anni più recenti.
La verifica delle oggettive uniformità del processo di automazione, al fine di poterne prevedere le principali linee di sviluppo, sia in termini quantitativi che qualitativi, ha costituito uno degli scopi principali dell'indagine qui riassunta. In altri termini, si sono cercati i fattori che hanno operato come cause del processo decisionale in materia di meccanizzazione con sistemi di elaborazione dati.
In proposito, si è condotta l'analisi su due linee di ricerca, dapprima cercando di inquadrare la spesa EDP nel contesto delle variabili caratterizzanti l'attività delle singole aziende di credito ([2]) e, successivamente, legando la stessa spesa alle manifestazioni dinamiche dei suddetti parametri.
Dai risultati emersi, comunque, non ci sembra ancora possibile parlare di una stabile relazione di struttura, né, tantomeno, di una legge di domanda EDP oggettivamente determinata del settore in esame.
Saremmo piuttosto portati a considerare il volume dei depositi, fattore che agisce in modo prevalente sulla manifestazione della domanda, solamente come un indicatore di prima approssimazione, in quanto tale variabile condizionerebbe, in media, soprattutto il livello di spesa in elaboratori.
In altri termini, la relazione tra depositi e spesa presenta ancora ampi margini di casualità (soprattutto all'interno di classi omogenee di aziende), che potrebbero derivare oltre che dalla rigidità della spesa anche da alcune componenti non ancora riconducibili



a schemi misurabili oggettivamente.
La situazione rilevata a fine 1973 (come installato più ordini netti) riguarda 208 sistemi per un valore di 2,6 milioni di punti ([3]).
Le nuove meccanizzazioni, insieme con i consistenti ampliamenti di centri già installati al 1968, hanno quasi triplicato, nel periodo 1968-73, la spesa dell'intero settore in elaboratori; volendo esprimere questo incremento in termini più sintetici, la spesa è aumentata ad un tasso medio annuo del 22,4% anche se alquanto differenziato per classi di deposito: tassi di sviluppo sostanzialmente decrescenti all'aumentare della dimensione aziendale (+45% della classe più piccola contro un modesto +16% della classe più alta) hanno ridotto il livello di concentrazione della spesa.
Considerando solamente la quota di incidenza sul totale delle aziende con oltre 300 miliardi di depositi, essa è scesa infatti dal 56% del 1968 al 42% del 1973.
Va comunque rilevato che la distanza tra il livello di spesa delle aziende più piccole e di quelle più grandi è alquanto ampia: dai 2500 punti medi per le aziende con meno di 20 miliardi di depositi, si passa ai 64800 punti della classe con più di 300 miliardi (Fig. 1 della Tabella C).
La mancanza di una relazione continua tra spesa e dimensione aziendale, per cui le banche più piccole devono affrontare la meccanizzazione a livello di spesa non proporzionale, emerge dalla considerazione del rapporto spesa/depositi per classe dimensionale delle aziende (Fig. 2 della Tabella C).
Ciò ci induce ad una ulteriore riflessione: l'incidenza della spesa sui depositi per le medie e grandi aziende tende a decrescere, non solamente perché si realizzano ovvie economie di scala, ma anche perché la domanda di queste aziende non si è adeguata, dal

([2]) Tra le possibili variabili esplicative del processo di meccanizzazione si sono considerati i seguenti gruppi:
1) Variabili caratterizzanti la dimensione operativa interna (numero di conti correnti, numero di depositi a risparmio mediamente esistenti all'epoca in cui è stata fatta l'intervista e numero di effetti commerciali e finanziari giacenti in caveau sempre allo stesso periodo).
2) Variabili caratterizzanti la frequenza dei rapporti con la clientela (numero medio di operazioni eseguite giornalmente sui conti correnti, numero medio di operazioni sui depositi a risparmio, numero medio di operazioni sui depositi a risparmio, numero medio giornaliero degli effetti presentati alla banca per l'incasso, sconto e simili).
3) Variabili caratterizzanti l'importanza economica dell'azienda sul mercato (numero sportelli, ammontare degli impieghi a breve termine, volume dei depositi).

([3]) Il punto rappresenta un dollaro di locazione mensile ed equivale ad una spesa annua di circa 8400 lire. Un parco di 2,6 milioni di punti rappresenta quindi una spesa annua di circa 22 miliardi di lire.

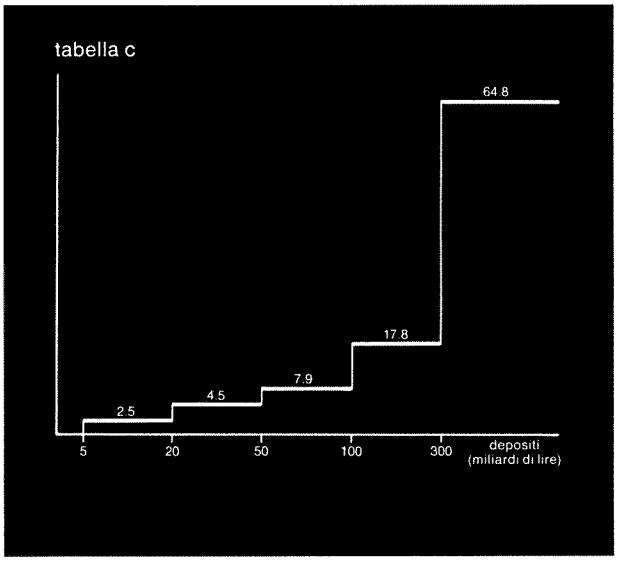


Fig. 1 - Spesa media EDP (in migliaia di punti).

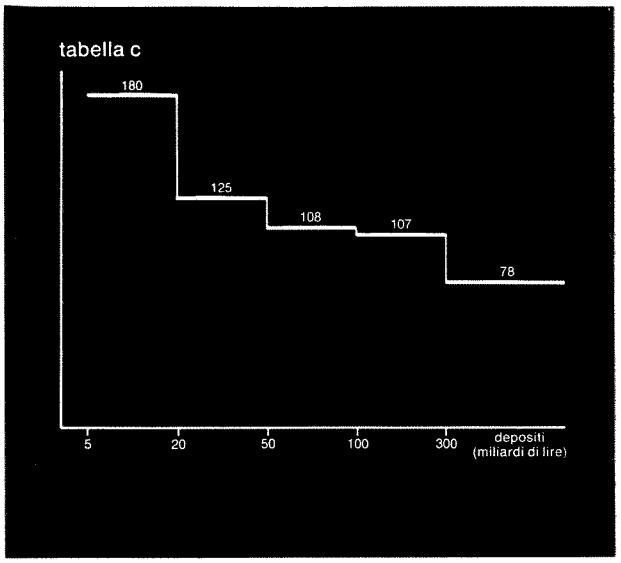


Fig. 2 - Punti per miliardo di deposito.

punto di vista applicativo, alle reali esigenze imposte della dimensione.

In altri termini, ci sembra che dall'indagine condotta emerga una tendenza delle aziende medio-grandi a completare le aree applicative di tipo tradizionale, ma non a sviluppare nuove aree applicative (programmazione strategica, ricerca operativa, ecc.).

Anche le applicazioni risultano, ovviamente, in stretta connessione con la dimensione del centro: per le piccole utenze prevalgono forme di automazione concentrate principalmente sui conti correnti, depositi a risparmio e portafoglio. Le altre procedure risultano invece meccanizzate quasi esclusivamente tra le utenze di più grande dimensione. Di particolare rilievo, fra queste, è la gestione dell'anagrafe clienti, con l'esistenza di un « central file » che permette una più alta integrazione del processo informativo.

Riteniamo che, per il settore globalmente considerato, alcuni limiti allo sviluppo del sistema informativo verso nuove e più sofisticate aree applicative, possano derivare anche dalle ridotte consistenze della Staff EDP del centro (Tabella D).

Oltre il 50% dei centri, infatti, non supera i 10 addetti in totale, ed ancora nella metà dei casi, risulta impiegato personale specializzato inferiore alle cinque unità.

Per quanto concerne le procedure meccanizzate dalle banche che operano in Service presso centri esterni, la maggioranza dei casi riguarda le operazioni sui conti correnti e sui depositi a risparmio, mentre più ridotto è il numero delle aziende che risultano avere meccanizzato il portafoglio e le procedure contabili.

Sotto il profilo delle caratteristiche tecniche, gli impianti EDP installati nel settore presentano una assoluta prevalenza delle configurazioni a dischi e a nastri/dischi (35% e 32% rispettivamente). I sistemi configurati a schede, sebbene ve ne sia ancora nel settore un certo numero (10% del parco), soprattutto presso le banche con un ammontare di depositi inferiore ai 20 miliardi, presentano sintomi di rapida so-

Fig. 1 - Personale complessivo dello staff.

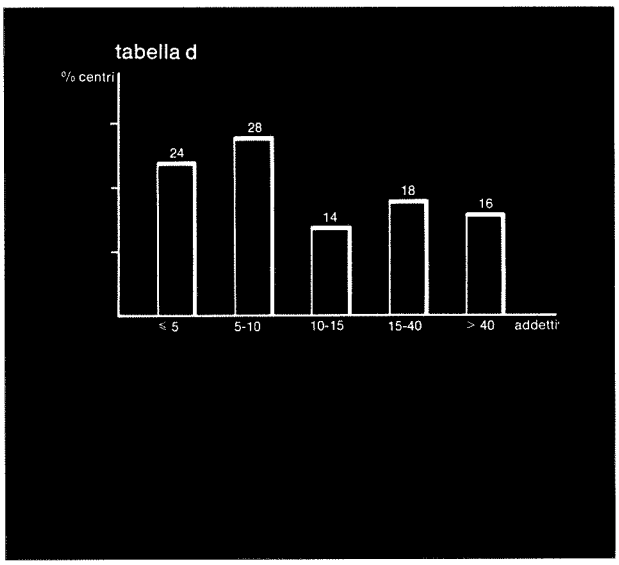
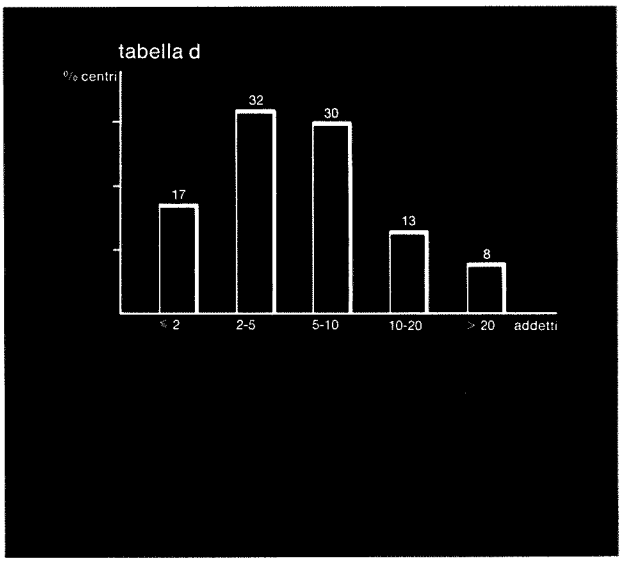
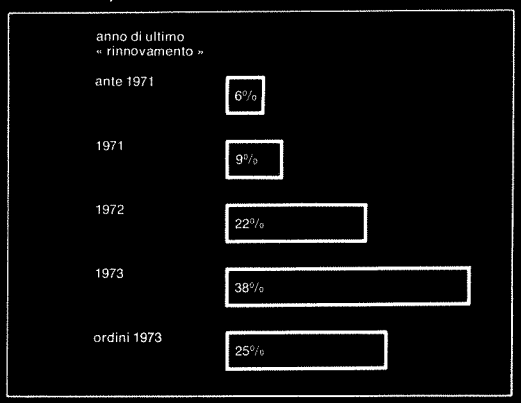


Fig. 2 - Personale specializzato dello staff.



stituzione; considerando gli ordini in essere al momento della rilevazione, la quota di questi impianti tende a diventare del tutto trascurabile.
Alquanto sostenuta si presenta poi la crescita della capacità della memoria: i sistemi in ordine presentano infatti una capacità più alta di circa il 25%.
In stretta connessione a questo sviluppo della memoria, possiamo considerare la situazione del « teleprocessing », cioè la diffusione di terminali per la gestione di procedure in tempo reale. Va rilevato a tale proposito che il 27% delle aziende utenti utilizza terminali on-line (la situazione si riferisce al parco installato più gli ordini netti); più in particolare, tale percentuale si avvicina al 50% per le aziende con più di 50 miliardi. Va precisato, inoltre, che la diffusione del tempo reale dovrebbe esprimere sviluppi particolarmente intensi, tenuto conto dei programmi rilevati in sede di indagine.
Infine, il dato di maggiore rilievo, emerso durante la ricerca sulla meccanizzazione EDP del settore, è rappresentato dalla dinamica di rinnovamento tecnico degli impianti.
In rapporto al parco installato, un quarto delle aziende ha effettuato ordini di « rinnovamento » del proprio centro. Comprendendo questi ordini, la quota dei centri, i cui impianti EDP non rinnovati presenta una anzianità superiore ai due anni, è pari al solo 15%.

tabella e - parco attuale secondo l'anno di installazione



Il materiale di questo articolo è stato tratto da uno studio effettuato dal Servizio Studi e Programmazione Marketing della Honeywell Information Systems Italia (V. Marandella, D. Rovina, « Indagine sul livello di meccanizzazione delle Casse di Risparmio e delle Banche Popolari Italiane », luglio 1974), cui si rimanda per ulteriori dettagli.

Resoconti e recensioni

Giornate di studio su « La programmazione strutturata »

Milano, 12-13 febbraio 1974. Organizzate dalla Honeywell Information Systems Italia in collaborazione con le Università di Bologna e Torino, con l'Università e Politecnico di Milano e con l'Istituto Elaborazione Informazione (CNR) di Pisa. Le relazioni sono in corso di pubblicazione sulla « Rivista di Informatica » dell'AICA.

Alle giornate, svoltesi presso il Centro di Formazione di Milano della Honeywell Information Systems Italia, sono state presentate 12 relazioni.

Dopo un breve richiamo dei « *problemi nella produzione industriale del software* » ad opera dell'ing. Cicu (direttore del Servizio Controllo Qualità Software presso il Centro di Ricerca e Progettazione della Honeywell Information Systems Italia, dove è in funzione una delle più grandi « software factories » oggi esistenti in Italia e l'unica specializzata in software di base), la relazione introduttiva *(« Che cos'è la programmazione strutturata »)* è stata svolta dal prof. Böhm dell'Università di Torino e dal prof. Degli Antoni dell'Università di Milano.

Sono seguite relazioni su: *« Tecniche di programmazione strutturata »* e *« Tecniche di programmazione strutturata in ambiente ASSEMBLER »* (M. Maiocchi, R. Polillo - Università di Milano); *« Un esempio di programmazione strutturata: il compilatore PASCAL »* (S. Crespi Reghizzi - Politecnico di Milano); *« Processi cooperanti »* (V. Montanari - Istituto Elaborazione Informazione, Pisa); *« Tecniche di programmazione strutturata di processi cooperanti »* (M. Boari - Università di Bologna e N. Lijtmaer, Istituto Elaborazione Informazione, Pisa); *« Definizione e gestione della struttura e dei dati di interfaccia di un grosso sistema di software »* (A. Cicu - Honeywell Information Systems Italia).

Al convegno è anche intervenuto Jean Dominique Warnier della Honeywell Bull di Parigi che ha presentato il metodo *« LCP: Leggi per la Costruzione dei Programmi »*, messo a punto da un gruppo da lui diretto e già utilizzato in parecchi paesi europei ed extraeuropei.

Nella sua formulazione più rigorosa la programmazione strutturata (che bandisce categoricamente la istruzione go-to per cui c'è anche chi ne parla come di « go-to-less programming ») prevede l'impiego, nella stesura di un programma, di tre sole strutture elementari chiamate rispettivamente *block, if-then-else* e *do-while* che rispondono rispettivamente alle funzioni di elaborazione sequenziale, decisione logica e iterazione. Un teorema, che va sotto il nome di Böhm e Jacopini, dimostra che qualunque programma può essere trasformato in un programma equivalente che faccia uso di queste sole tre strutture. Per evitare tuttavia possibili appesantimenti e perdite di efficienza altre due strutture vengono ammesse oltre alle tre di base e cioè la struttura *case* per la funzione di scelta fra più alternative e la struttura *do-until* che è la struttura iterativa inversa a do-while.

Due caratteristiche, fondamentali per gli scopi che la programmazione strutturata si propone (e cioè la facilità di controllo e « manutenzione » di un programma), sono il fatto che ogni struttura presenta un solo punto di entrata e un solo punto di uscita e il fatto che le strutture devono essere impiegate in un contesto di programmazione di tipo « top-down », cioè partendo dal livello più elevato e procedendo a definire successivamente i livelli gerarchicamente inferiori.

La programmazione strutturata (di cui il metodo Warnier o LCP rappresenta una delle prime applicazioni coerenti a livello didattico) si situa in un processo di razionalizzazione e standardizzazione logica che finora aveva interessato soprattutto gli aspetti hardware dell'informatica e in particolare quelli circuitali e che, grazie ad essa, viene esteso agli aspetti software, e quindi anche alla formazione degli analisti/programmatori.

« Quaderni di informatica » si riserva di tornare ampiamente sull'argomento della programmazione strutturata in uno dei prossimi numeri.

(G. R.)

Giornata di studio su « L'inserimento della affidabilità nei contratti. Metodi ed esperienze nel settore degli apparati elettronici »

Milano 29 Novembre 1973, organizzata dall'Associazione Nazionale Italiana per l'Automazione (ANIPLA) e dalla Associazione Italiana per il Controllo di Qualità (AICQ).

La giornata ha rappresentato l'occasione per fare il punto su un problema complesso e delicato come quello delle garanzie contrattuali sulla affidabilità nel campo delle apparecchiature elettroniche. Sono state presentate metodologie ed esperienze riguardanti i settori principali, dalle telecomunicazioni agli elaboratori elettronici. Per quest'ultimo settore, relatore è stato l'ing. G. Moroni della Honeywell Information Systems Italia.

Egli ha anzitutto rilevato che le richieste relative alla affidabilità, in precedenza limitate ad utenti militari od enti statali, cominciano ora ad avere una certa generalizzazione.

Ciò è dovuto ad una migliore conoscenza e diffusione delle tecniche di affidabilità, a maggiori esigenze operative e ad una aumentata sensibilità manageriale.

E' stato quindi affrontato il problema del significato e della misura delle prestazioni di un calcolatore, elementi indispensabili per poter fissare dei parametri oggettivi a fronte dei quali far valere una garanzia. Il problema non è semplice in quanto il livello di efficienza del sistema — il fattore che è noto in letteratura tecnica come la « system effectiveness » — dipende in modo complesso e difficilmente generalizzabile dalla affidabilità dei singoli componenti il sistema, dal grado di ridondanza che il sistema presenta a fronte di determinati tipi di guasto e infine dal livello di « riparabilità » dei singoli guasti e dalla efficienza della organizzazione di manutenzione.

Pertanto, ci si limita molte volte a dare garanzie numeriche al parametro « disponibilità », che in pratica è espresso come rapporto tra il tempo utile di funzionamento e la somma del tempo utile più il tempo in cui il sistema non è in grado di fornire il servizio richiesto perché in attesa di essere riparato o perché l'intervento di manutenzione è in corso.

La « disponibilità » è un numero teoricamente compreso tra 0 e 1 e per i normali sistemi di calcolo varia tra 0.97 e 0.99.

(M.D.M.)

Stato e automazione

di Mario G. Losano, pagg. 245, Etas Kompass Libri 1974.

Polarizzata com'è l'attenzione degli informatici italiani da quanto succede di là dall'Atlantico, scarse notizie si hanno in Italia della situazione in materia EDP anche dei paesi vicini. Figuriamoci poi per un paese per tutti i versi così distante da noi come il Giappone. A proposito giunge dunque questo volume che, pur portando un titolo generico, all'esperienza giapponese interamente si riferisce. E non si tratta di un'esperienza di poco conto, visto che il Giappone anche in campo EDP (e cioè come numero di elaboratori elettronici installati e prodotti) è il secondo paese del mondo dopo gli Stati Uniti.

Mossosi con l'intento di esaminare l'automazione della pubblica amministrazione giapponese, molto opportunamente l'autore ha ritenuto necessario « non solo studiare il movimento che dall'industria va allo stato-utente, ma anche completare il cerchio, studiando il movimento che dallo stato-incentivatore va all'industria ». Del resto una delle più efficaci forme di incentivazione, in Giappone come in altri paesi, è proprio costituita dal privilegiare l'industria nazionale come fornitrice di elaboratori elettronici allo stato e agli enti pubblici che in tutti i paesi rappresentano una quota molto importante della domanda EDP.

A questo studio viene premessa una introduzione storica molto utile per capire la realtà socioeconomica di un paese che non ha conosciuto la rivoluzione borghese né quindi il liberismo economico ma ha direttamente « trasferito una struttura feudale nel mondo della rivoluzione industriale ». Da ciò il permanere di « un fortissimo senso delle istituzioni intermedie » che fa delle aziende attuali le dirette eredi dei clan medievali. E da ciò quella straordinaria interpenetrazione di pubblico e privato, quel peculiare rapporto fra stato e aziende « che fa del Giappone anche odierno un'unica, colossale impresa produttiva che abbraccia burocrati, operai, diplomatici e imprenditori », tanto che gli americani si divertono a parlare di una « Japan Inc. ».

Anche per quanto riguarda l'industria dei calcolatori, l'intreccio fra azione pubblica e azione privata è fin dall'inizio quanto mai singolare, almeno agli occhi di un occidentale. I primi calcolatori giapponesi nascono presso il Laboratorio Elettrotecnico del Ministero per l'industria e il commercio estero, nel 1956/57. Subito dopo però lo stato « passa la mano » all'industria privata e in particolare ai grandi gruppi già operanti nel campo delle telecomunicazioni e dell'elettronica professionale e di consumo (Hitachi, Fujitsu, NEC, Toshiba ecc.) spingendoli a ricercare accordi tecnici con l'industria americana (e questo non per una pretesa inferiorità tecnologica o per la tendenza a « copiare » generalmente attribuita ai giapponesi, ma per guadagnare tempo). Contemporaneamente, con la prima legge di incentivazione per l'industria elettronica del 1957, lo stato finanzia e al tempo stesso coordina, per evitare duplicazione di sforzi, l'attività industriale e di ricerca delle varie case. Inoltre promuove all'inizio degli anni 60 la costituzione della Japan Electronic Computer Company (JECC) che, nei confronti delle industrie nazionali, finanzia con fondi statali il noleggio degli elaboratori (l'utente sceglie l'elaboratore, la JECC lo

compra e lo paga alla casa costruttrice e lo noleggia all'utente). Tutto ciò, accompagnato da varie misure protezionistiche, ha fatto sì che la presenza diretta dell'industria americana (IBM, NCR, Borroughs e Univac) installatasi subito dopo la guerra sia rimasta nei limiti di una posizione di mercato minoritaria. Dovendo ora abbandonare per accordi con gli Stati Uniti quelle misure protezionistiche, per rafforzare ulteriormente la posizione delle industrie nazionali (che non hanno mancato nel frattempo di « farsi le ossa » sino a rendersi in alcuni casi autonome anche sul piano tecnico e ad affacciarsi con propositi di conquista sui mercati del terzo mondo) lo stato giapponese ha varato una nuova legge di incentivazione e ha promosso una vera e propria « fusione operativa » a due a due fra le sei case produttrici di elaboratori « general purpose » (Fujitsu-Hitachi; NEC-Toshiba; Oki-Mitsubishi).

Tutto ciò riguarda il lato dell'offerta. Ma lo stato giapponese è assai attivo anche per quanto riguarda la domanda EDP e non solo quella rappresentata dagli uffici pubblici (per i quali, come già accennato, vale la regola della preferenza da accordare negli acquisti alle industrie nazionali). Sotto auspici governativi il CUDI (Japan Computer Usage Development Institute) ha preparato due piani, uno a media e uno a lunga scadenza, per l'« informatizzazione della società », e qualcuno ha già parlato di un possibile privilegiamento dello sviluppo dell'informatica in Giappone paragonabile al privilegiamento avutosi fino ad epoca molto recente in molti paesi per la motorizzazione. E va notato, come fa il Losano, che se questo sviluppo è favorito dalla notevole conformità sociale e dall'omogeneità culturale caratteristiche del Giappone, ad esso si contrappongono difficoltà locali che non hanno riscontro in Occidente, come il problema (ben illustrato dall'autore) del doppio alfabeto, fonetico e ideografico.

I due piani di « informatizzazione » si riferiscono fra l'altro alla razionalizzazione con strumenti EDP della pubblica amministrazione, all'evoluzione del « management information system » nelle aziende, all'istruzione con l'aiuto dell'elaboratore, alla modernizzazione delle cure mediche, al controllo e alla prevenzione dell'inquinamento, all'automazione del controllo del traffico e alla modernizzazione dei canali di distribuzione. Essi comprendono anche il progetto di una « Computepolis », ossia di una città computerizzata, e prevedono non lontana una progressiva diffusione dei terminali a domicilio (già nel 1985 l'uno per cento delle famiglie giapponesi dovrebbe avere in casa il terminale).

Il Losano, oltre ad avvertire che questi piani sono stati formulati prima del sopravvenire della « crisi energetica » particolarmente grave — com'è noto — per il Giappone, parla — almeno per taluni aspetti di essi — di « computopia ». Ma va detto che, mentre la prima parte del libro (quella che si riferisce all'industria giapponese dei calcolatori e ai rapporti industria-stato) è ammirevole per ricchezza di informazione e chiarezza di esposizione, la seconda parte (che è poi quella da cui il libro ha preso le mosse e in cui vengono esposte alcune applicazioni avanzate di elaboratori nella pubblica amministrazione giapponese e, appunto, i piani di « informatizzazione ») avrebbe meritato un maggiore sviluppo e un adeguato approfondimento critico della materia.

(G. R.)

Implicazioni sociali dell'uso di elaboratori attraverso i confini nazionali

(The Social Implications of the use of Computers across National Boundaries).
Rapporto al « Comitato per le implicazioni sociali degli elaboratori » della American Federation of Information Processing Societies, AFIPS, Nov. 1973, pgg. 145.

Negli ultimi anni si è avuto un crescente interesse per gli aspetti sociali dell'informatica. L'esperienza ha messo in luce che il progresso della scienza e della tecnologia è spesso accompagnato da effetti collaterali negativi che, una volta manifestatisi, sono difficili da controllare; a questo proposito è sufficiente citare l'inquinamento dell'ambiente.

Nel settore dell'informatica si è cercato di essere più previdenti; infatti fin dal 1970 una nota personalità nel settore dell'informatica, Louis Feldrer, rivolgendosi ai membri del Congresso U.S.A., raccomandava l'istituzione di un sistema di « osservazione e di allarme » per permettere al potere politico di conoscere con adeguato anticipo le conseguenze della diffusione e delle applicazioni degli elaboratori. Su questi argomenti si è sviluppata una vasta letteratura e si sono svolti numerosi convegni e dibattiti. In questo filone si colloca lo studio dell'AFIPS (American Federation of Information Processing Societies) intitolato « The Social Implications of the use of Computers across National Boundaries ». L'esigenza di uno studio di tal genere è stata stimolata dalla carenza di investigazioni specifiche sull'argomento; d'altra parte il crescente « multinazionalismo » delle aziende, insieme con lo sviluppo della trasmissione dati, il miglioramento delle prestazioni degli elaboratori, l'avvento delle reti di elaborazione, lasciano prevedere un numero rapidamente crescente di sistemi di elaborazione composti da elementi collocati in differenti nazioni.

La metodologia seguita nella ricerca è nota sotto il nome di « Delphi » e fu introdotta per prima dalla Rand Corporation. Si tratta di un metodo per integrare, in modo sistematico, le opinioni di un gruppo di esperti selezionati dal mondo industriale, scientifico e accademico. Allo studio in questione, ha partecipato, per l'Italia, il dr. Franco Filippazzi, Direttore della Ricerca della Honeywell Information Systems Italia.

Le conclusioni del gruppo di esperti sono raccolte in un voluminoso fascicolo che analizza i vari aspetti del problema. Ecco alcune delle indicazioni più significative:

Il 50% degli esperti interrogati ritiene che uno degli aspetti più significativi da esaminare è il controllo delle maggiori aziende multinazionali da parte dei paesi più ricchi. L'adozione di sistemi multinazionali di elaborazione dei dati accrescerebbe ulteriormente la potenza economica di queste organizzazioni aumentando il divario tra i paesi più ricchi e quelli più poveri.
Il 56% degli esperti interrogati ritiene molto importante l'aspetto del contributo all'omogeneizzazione dei valori culturali nei vari paesi, tenendo però presente che ciascun paese potrà seguire una via diversa per raggiungere l'omogeneizzazione e talvolta questo processo potrà dar luogo a turbamenti sociali.
L'83% degli esperti interrogati ritiene che esistono delle barriere ai sistemi multinazionali di elaborazione dei dati e questi ostacoli siano costituiti da conflitti ideologici e da un certo modo di intendere la sovranità nazionale.
Il 59% degli esperti interrogati ritiene che esista una carenza di meccanismi politici e legali per risolvere controversie che potrebbero sorgere sulle banche dei dati.
Il 56% degli esperti interrogati ritiene che la eventuale tendenza da parte di alcuni paesi a porre ostacoli al commercio internazionale dei calcolatori non influirà sullo sviluppo dei sistemi multinazionali di elaborazione dei dati.
Scopo dello studio non era soltanto la previsione di problemi che potrebbero sorgere in seguito all'uso di reti multinazionali di calcolatori, ma anche di esprimere un giudizio su alternative proponibili. E' interessante notare come viene considerata utopistica e non probabile l'ipotesi che l'uso di reti multinazionali di calcolatori possano ridurre il divario tra i paesi più ricchi e quelli più poveri, mentre si preconizzano conflitti di interesse tra le organizzazioni che utilizzeranno le reti multinazionali e gli stati in cui esse opereranno.
Infine viene esaminato l'ambiente sociale, tecnologico, culturale ed organizzativo in cui opereranno le reti multinazionali ed adeguato rilievo è dato anche alle considerazioni politiche.
Il comitato di esperti ha dato inoltre una serie di suggerimenti su altre aree di ricerca nell'ambito dell'influenza dei sistemi di elaborazione sull'ambiente sociale, culturale, economico e politico.
Lo studio è corredato da una ricca bibliografia, un elenco dei sistemi multinazionali di elaborazione di dati attualmente esistenti ed infine i risultati analitici ottenuti con il metodo Delphi.

(M.D.M.)

Cibernetica e modelli sociali

R.M. Page - Some Models in Group Cybernetics, Honeywell Computer Journal, Vol. 7, n. 2, 1973.

L'autore definisce « Cibernetica di Gruppo » l'estensione della Cibernetica allo studio di gruppi di individui, funzionanti come unità per effetto dei propri sistemi di comunicazione e dei controlli automatici che generano un comportamento caratteristico. Prima di trattare la vera e propria cibernetica di gruppo egli considera, in una scala di crescente complessità, i diagrammi a blocchi che rappresentano, da un punto di vista cibernetico, le macchine, gli animali, e gli uomini.
La fig. 1 fornisce lo schema a blocchi estremamente semplificato, di un elaboratore di processo.
L'animale (fig. 2) può essere considerato come un calcolatore di processo in cui, al posto del programmatore esterno, vi è una memoria a « sola lettura » in cui sono iscritti programmi fissi, e cioè gli istinti. Inoltre l'unità di Azione è anche controllata direttamente dai dati in ingresso (azioni riflesse) e tanto l'unità di azione, che i risultati, forniscono dati e « routines » da registrare nella memoria di lettura-scrittura (processo di apprendimento).
L'uomo, secondo l'autore, può essere rappresentato da un diagramma più complesso, comprendente una parte sostanzialmente uguale a quella di un animale (« corpo ») e una parte caratteristica dell'uomo (« psiche »). L'autore assume che, in questa parte caratteristica, esista un operatore-programmatore incorporato (e cioè la « libera volontà »), capace di imporre le sue decisioni a tutti gli altri componenti decisionali ed operativi, indipendentemente dalle « routines » fisse (istinti) e dai dati in memoria. La memoria di lettura-scrittura, oltre alla parte comune all'animale, comprende una parte caratteristica dell'uomo che (semplificando le indicazioni date dall'autore) contiene ciò che si può definire la sua « personalità » (concetti, norme morali, ideali estetici, ecc.) che, nel suo insieme, si può paragonare al « programma supervisore » di un calcolatore. Essa inoltre è connessa a una particolare unità di calcolo, l'« intelligenza » che, in cooperazione con la « personalità » e sotto il controllo della « libera volontà » elabora dei « criteri di validità » o « giudizi », che hanno una certa analogia con le « routines » diagnostiche, per la rivelazione e la correzione degli errori. Come si vede dalla fig. 3, la linea tratteggiata orizzontale indica la divisione statica dell'uomo in « corpo » e « psiche », mentre la molteplicità dei collegamenti verticali sottolinea l'unità dinamica delle due componenti.
A questo punto l'autore passa a considerare gruppi di individui, e le loro reciproche relazioni di comunicazione e di controllo, che ne fanno unità cibernetiche. In fig. 4 viene considerata una società animale (gregge, stormo, branco, ecc.) di individui della stessa specie, aventi quindi istinti uguali e fissi. Il diagramma è analogo a quello di una semplice calcolatrice, ma anziché essere comandato da un operatore, il gruppo è governato da una memoria a sola lettura, sede degli istinti del gruppo, che sono il risultato di quelli degli individui. Perciò la società animale è assai forte e stabile, tutte le leggi essendo rigidamente rispettate. Per quanto riguarda le società umane, si possono co-

Fig. 1 - Elaboratore di processo.

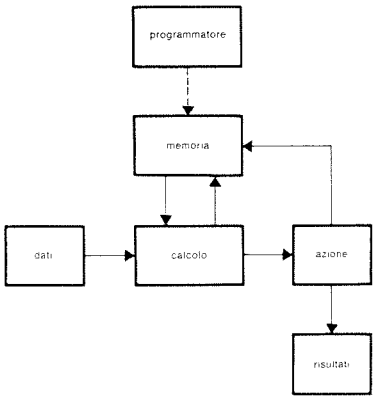


Fig. 2 - L'animale come elaboratore di processo.

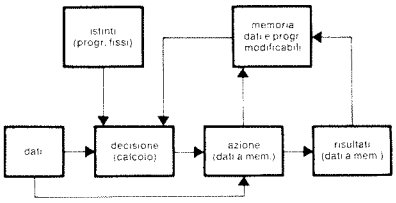


Fig. 3 L'uomo.

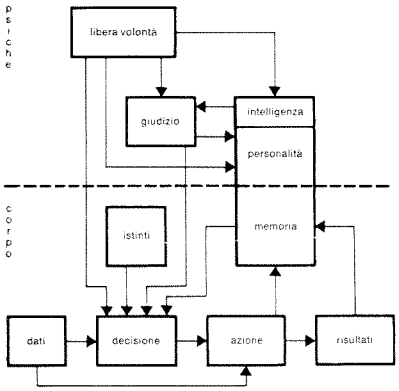


Fig. 4 La società animale.

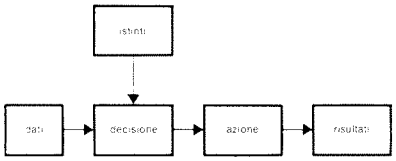


Fig. 5 La società assembleare.

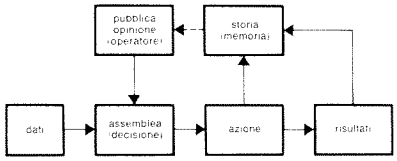
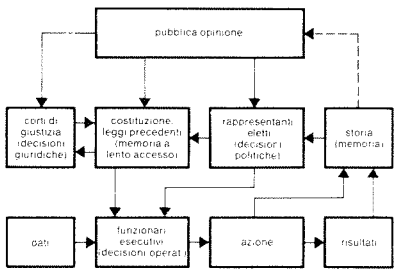


Fig. 6 La repubblica costituzionale.



struire diversi modelli. Una società governata da un potere assoluto è simile a una calcolatrice: in essa, al posto dell'operatore v'è il « capo » (monarca assoluto, dittatore, tiranno, ecc.). Nelle società in cui il governo è in maggiore o minor misura influenzato dal consenso dei governati, la libera volontà di questi viene riconosciuta, e, più o meno, rispettata.

Delle varie società di questo tipo, l'autore considera, a titolo di esempio, il modello di quella che egli chiama « democrazia pura », e che è nota, più propriamente, come « democrazia assembleare ».

Secondo lo schema indicato in fig. 5, la pubblica opinione, considerata come la somma delle libere volontà dei cittadini, influenza l'unità di decisione, che è l'assemblea, e che determina l'azione, in base ai dati di ingresso. Esiste in questo modello una memoria, che registra la « storia » dei diversi risultati successivamente conseguiti. Tale memoria non ha azione diretta sulle decisioni, ma influenza (o dovrebbe influenzare) la pubblica opinione, e di qui le decisioni. La chiusura dell'anello rende lo schema simile a quello di un elaboratore, con la differenza che l'operatore è incluso entro l'anello di « feed-back ». La memoria, ossia la cognizione della storia, dovrebbe avere una influenza stabilizzante: senza di essa, la variabilità della pubblica opinione, influenzata direttamente dai risultati, e influenzante direttamente le decisioni, può condurre a pericolose condizioni di instabilità.

Il secondo modello esaminato dall'autore è quello che egli chiama « repubblica costituzionale ». In essa si introducono nuovi elementi che ne fanno un insieme di elaboratori interlacciati. Vi sono « Corti di giustizia » che producono decisioni giuridiche, « Camere di Rappresentanti Eletti » che producono decisioni politiche, e « Funzionari esecutivi » che debbono realizzare tali decisioni. Connessa con questi organi, l'autore considera una « Memoria a lento accesso », distinta dalla memoria storica, che contiene la Costituzione con i suoi successivi emendamenti, le leggi, i precedenti giuridici. Le decisioni politiche diventano leggi attraverso ad essa, ed essa pone vincoli alle decisioni dei funzionari esecutivi. La pubblica opinione non può agire direttamente sulle decisioni, ma o le influenza indirettamente attraverso i rappresentanti eletti, o modifica il corpo delle leggi attraverso le corti di giustizia e gli emendamenti alla costituzione.

Questa mancanza di connessione diretta fra risultati, pubblica opinione e decisioni esecutive è, a giudizio dell'autore, un fattore di stabilità, che tuttavia permette un continuo controllo della pubblica opinione sul governo, nel rispetto delle libertà civili.

Una facile obiezione che sorge alla lettura di questo lavoro è che la realtà socio-politica è ben altrimenti articolata e complessa degli schemi informatici proposti dall'autore. Si può tuttavia riconoscere a questa analisi un valore provocatorio, di stimolo intellettuale.

(D.G.)